EXBASIC LEVEL II™

CBM 2000 · 3000 · 4000 · 8000

```
30 LETTEROFF:FORA=0TO3:READR(A):NEXT:C$=STRING$(40,"-"):S=826
40 READA$:IFA$<>"@"POKES,DEC(A$):S=S+1:GOTO.
50 F$="######":Y=151:S=32:L=32:U=120:T=33167:V=81:X=50:Z=74:Q=
60 M=17100:D=1:P=0:O=32967:E=207
70 D=D-1:H=40:C=33708
80 PRINTUSING"▌ ▚BALL▀ # ▚HIGHSCORE▀ "+F$+" ▚POINTS▀: "+F$,D,A
90 PRINT"╭"STRING$(38,"-")"╮";
100 A$="|▚┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬▀|":PRINTA$A$A
110 B$="|"+STRING$(38,32)+"|":PRINTB$B$A$A$A$B$B$A$A$A$;
120 FORI=1TO8:PRINTB$;:NEXT:PRINT"|"STRING$(18,32)"▔▔"STRING$
130 PRINT"╰────────╮"STRING$(16,".")"╭────────╯▮▮▮─";:POKE18
140 IFNOTGPRINT@726,USING"▚GAME OVER        SCORE="+F$,SC
150 D=D+1:PRINT@5,D:A=33649+RND(38):R=-39:F=1
160 A=A+R:B=PEEK(A):IFB=SPOKEA,V:GOTO270ELSEIFB<ETHEN200
170 F=3-F:POKEA,S:IFB=EPOKEA+1,S:A=A+1ELSEPOKEA-1,S
180 H=H-.1:P=P+X:IFA<TTHENP=P+X:IFA<OTHENP=P+X
190 PRINT@33,USINGF$,P:IFP<MTHEN220ELSEM=M+17100:GOTO70
200 IFB<>UTHEN230
210 A=A-Q:PL=PEEK(Y)AND3:IFPL=2F=0ELSEIFPL=1F=1ELSEF=3-F
220 R=R(F):GOTO160
230 IFB=46POKE158,0:IFD=5THEN310ELSEWAIT158,1:GETA$:GOTO140
240 IFB=64F=3-F:GOTO220ELSEIFB=93F=5-FAND3:GOTO220
250 IFB<128F=F+2AND3:GOTO220ELSEIFF<2THEN220ELSEPOKEC-1,S
260 POKEC,S:POKEC+1,S:C=A+4*R:POKEC-1,U:POKEC,U:POKEC+1,U:GOTO
270 IFGFORI=0TOH:NEXT:GOTO290
280 IFPEEK(Y)=ZTHENG=-1:PRINT@726,STRING$(28,32):DOKE144,826:G
290 IFPEEK(A)<>UPOKEA,S
300 GOTO160:DATA-41,-39,41,39
310 SC=P:SEC1:IFP<A(5)THEN370
320 R=1:FORI=5TO1STEP-1:IFA(I)>PTHENR=I+1:I=0
330 NEXT:OFF:PRINT"▌"C$:PRINT"    ▚GAME OVER▀   SCORE:"P:PRINT
340 PRINTUSING"▚YOU ARE ON THE #. PLACE!",R
350 FORI=5TORSTEP-1:A$(I+1)=A$(I):A(I+1)=A(I):NEXT:A(R)=P
360 INPUTFORM"▚ENTER YOUR NAME:  ";A$(R),15
370 PRINT"▌"C$"▚THE FIVE ▚BESTS▀ ARE:▚▚
380 FORI=1TO5:PRINTA$(I)TAB(20)A(I):NEXT:PRINTC$:G=0
390 FORI=1TO1E3:GETA$:IFA$="▚"DISPOSENEXTELSENEXT
400 GOTO50:DATAA5,8F,29,02,D0,2D,A6,BD,A5,97,30,27,4A,90,11,E6
410 DATAE8,A9,20,9D,99,83,A9,78,9D,9C,83,D0,11,4A,90,10,8A,F0,
```


Andreas Dripke   Michael Krause



EXBASIC LEVEL II™

EXBASIC LEVEL II

Ein Produkt von

Michael Krause

und

Andreas Dripke

Vertrieb durch

INTERFACE AGE
Verlagsgesellschaft

EXBASIC LEVEL II
Made in West Germany

Gesamtherstellung:
Unternehmensberatung Andreas Dripke, Wiesbaden

Druck der Anleitung: Offsetdruck Kapehl, Wiesbaden

EXBASIC LEVEL II™

EXBASIC LEVEL II Benutzerhandbuch

1. Auflage in deutscher Sprache: 1981
2. Überarbeitete Auflage in deutscher Sprache: 1982

Vorwort

Die Nutzbarkeit eines Computers steht und fällt mit der Software - eine Erkenntnis, die schon viele Anwender aus leider meist negativen Erfahrungen gewinnen mußten. Dabei ist für denjenigen, der seinen Computer überwiegend selbst programmiert, das Betriebssystem und die damit verbundene(n) Programmiersprache(n) von besonderer Wichtigkeit. BASIC ist die unter Mikrocomputern am meisten verbreitete Programmiersprache, da sie besonders leicht zu erlernen und einfach anzuwenden ist. Diese Vorzüge werden aber im allgemeinen mit einem sehr eingeschränkten Wortschatz erkauft. Mit anderen Worten - die Leistungsfähigkeit der meisten BASIC-Dialekte läßt sehr viel zu wünschen übrig, so auch das Commodore BASIC. Das bedingt, daß die Programme lang, unübersichtlich und oftmals mit so vielen Tricks und Kniffen versehen werden, daß nach kurzer Zeit selbst der Programmierer nicht mehr sein eigenes Programm versteht; ganz abgesehen von dem enormen Zeit- und Arbeitsaufwand, der mit solchem ineffizienten Programmieren verbunden ist.

Aus diesen Erfahrungen heraus entwickelte sich bei vielen Anwendern von Commodore Computern der Wunsch nach einer leistungsfähigeren Programmiersprache, die aber auf dem normalen BASIC aufbauen sollte, damit das einmal Gelernte nicht völlig über Bord geworfen werden muß. Es erfüllt uns mit Freude und Stolz, Ihnen dieses stark erweiterte und äußerst leistungsfähige BASIC für Commodore Computer zur Verfügung stellen zu können - es heißt EXBASIC LEVEL II.
EXBASIC LEVEL II stellt eine - wie wir meinen - ideale Kombination des normalen Commodore BASIC mit dem weltweit anerkannten LEVEL II BASIC Standard dar. Hinzu kommt noch eine geradezu unglaubliche Fülle von weiteren Funktionen, die Ihnen EXBASIC LEVEL II ("Extended Level II Basic", "Erweitertes Level II Basic") bietet - angefangen von den schon vom Toolkit her bekannten Hilfsfunktionen über Mathematik- und Graphikbefehle, FAST TAPE, DOS SUPPORT, SCREEN SUPPORT und EXMON bis hin zur Erweiterung mit SOFTMODULEN.

Damit Sie die vielfältigen neuen Möglichkeiten auch richtig nutzen können, haben wir dieses Handbuch geschrieben. Wir werden ausführlich jede neue Funktion erklären und anhand von Beispielen besprechen. Allerdings erwarten wir dabei von Ihnen Grundkenntnisse des normalen Commodore BASIC. Sollten Sie noch nicht soweit sein, so empfehlen wir Ihnen in die Anfangsgründe des Commodore BASIC einzutreten mit dem ebenfalls bei INTERFACE AGE (COMPUTER LIFE) erschienenen 'CBM/VC-20 BASIC-BUCH für Beginner'.

Nun wollen wir Sie nicht länger "auf die Folter spannen", sondern uns einer der besten Programmiersprachen der Welt zuwenden. Wir wünschen Ihnen viel Erfolg mit EXBASIC LEVEL II.

Inhaltsverzeichnis

1. Einleitung 7

2. EXBASIC LEVEL II Befehlssyntax 11

3. Hilfsfunktionen 13

4. Graphikbefehle 27

5. Mathematische Funktionen 33

6. LEVEL II BASIC Befehle 37

7. Fehlermeldungen 69

8. Hinweise 71

9. Kassettenrekorderbefehle (CBM 2001/3001/4001) .. 73

10. Floppykurzbefehle (CBM 2001/3001/4001) 77

11. Bildschirmsonderbefehle (CBM 8001) 79

12. Assembler/Disassembler (CBM 8001) 85

13. SOFTMODULE 89

14. Beispielprogramme in EXBASIC LEVEL II 95

15. ANHANG: Befehlsliste 99

16. Notizen 105

17. Stichwortverzeichnis 115

Einleitung

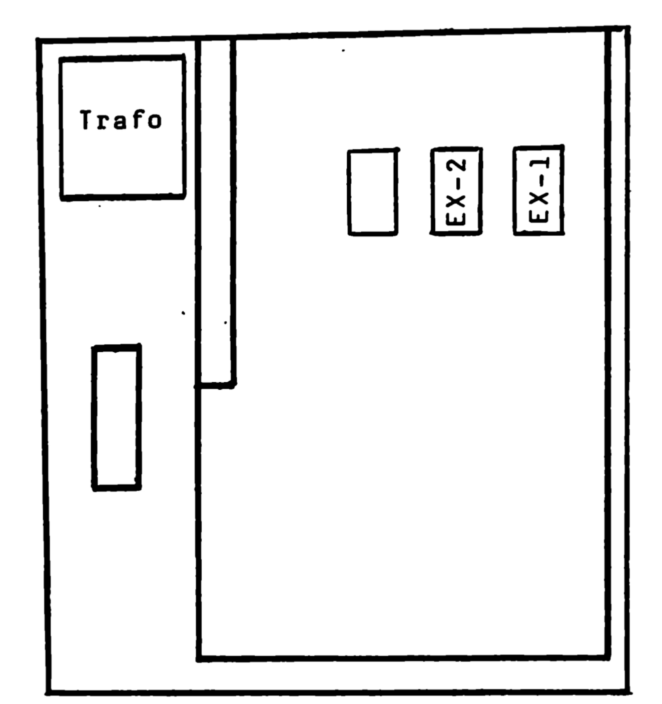
EXBASIC LEVEL II stellt ein stark erweitertes Basic für alle Commodore Computer der Serien 2001 (nur mit neuen Roms!), 3001, 4001 und 8001 dar. Es belegt 8 kByte Speicherkapazität zwischen 36864 (9000 hex) und 45055 (AFFF hex). Im Computer sind für diese Erweiterung bereits zwei freie Stecksockel vorgesehen. Dies bedeutet zum einen, daß Sie EXBASIC LEVEL II durch einfaches Einstecken selbst implementieren können und zum anderen, daß die 8 kByte, die EXBASIC LEVEL II belegt, nichts vom Basic Speicherbereich abziehen. Ihnen steht also auch weiterhin die volle Speicherkapazität Ihres Computers zur Verfügung.

Den Fachleuten sei gesagt: EXBASIC LEVEL II benötigt einige Bytes in der Zero-Page sowie aus dem ersten Kassettenpuffer. Dieser kann aber trotzdem weiterhin ohne Einschränkung benutzt werden. Der zweite Kassettenpuffer wird nicht belegt und steht somit weiterhin für Maschinenprogramme zur Verfügung.

Zunächst einmal wollen wir die beiden Bausteine (es handelt sich um sog. Eproms) von EXBASIC LEVEL II in unseren Computer einsetzen. Dazu ist es notwendig, den Computer erst einmal zu öffnen. ACHTUNG: ZIEHEN SIE VORHER UNBEDINGT DEN NETZSTECKER!!!
Lösen Sie, möglichst mit einem passenden Kreuzschlitzschraubendreher, zur Not tut es aber auch ein normaler, die Schrauben rechts und links unterhalb der Abdeckplatte etwa in Höhe der Tastatur. Drehen Sie die Schrauben ganz heraus und legen Sie sie nebenhin. Sollten sie ausnahmsweise so fest sitzen, daß sie nur schwer zu lösen sind, so verwenden Sie unbedingt einen passenden Kreuzschlitzschraubendreher weil Sie andernfalls den Kreuzschlitz so stark beschädigen könnten, daß die Schraube dann überhaupt nicht mehr herauskommt!

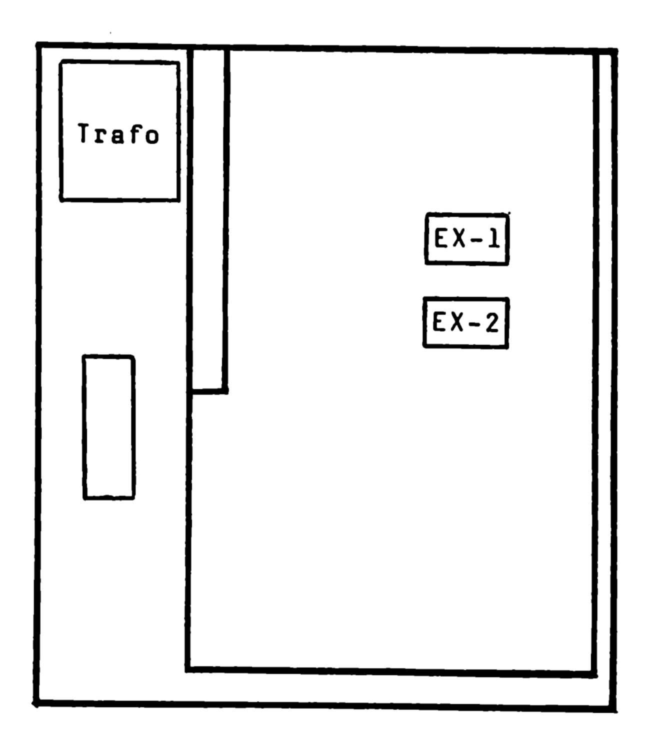
Wenn die Schrauben gelöst sind, klappen Sie den Computer vorsichtig nach hinten auf. Vorne oder an einer Seite finden Sie einen Feststellarm, mit dem Sie den Computer offenhalten können. Nun sollten Sie für eine gute Beleuchtung des Computerinnern sorgen. Positionieren Sie den Computer am besten an einem Fenster oder stellen Sie eine gute Schreibtischlampe daneben.
Packen Sie nun Baustein 1 von EXBASIC LEVEL II vorsichtig aus. Berühren Sie diese empfindlichen hochintegrierten Bausteine möglichst nur am Gehäuse und nicht an den blanken Anschlußpins. Bedenken Sie auch, daß diese Bausteine etwas gegen elektrostatische Aufladungen haben. Setzen Sie nun Baustein 1 vorsichtig und leicht auf den in nachfolgender Skizze bezeichneten freien Sockel auf. Die Richtung ergibt sich ebenfalls aus der Skizze, stellen Sie unbedingt sicher, daß der Baustein nicht falsch herum liegt! Überprüfen Sie jetzt sorgfältig (!), ob alle Anschlußpins auch richtig aufsitzen. Erst dann drücken Sie den Baustein vorsichtig in den Sockel ein.
Verfahren Sie danach analog mit Baustein 2.

Diese Skizzen zeigen, wie die beiden zu EXBASIC LEVEL II gehörenden Bausteine in die dafür vorgesehenen freien Sockel des Computers eingesteckt werden.



Die hier abgebildeten Sockel sind normalerweise frei, das heißt, in ihnen steckt kein schwarzer Baustein.

Je nachdem, ob in dem Computer die freien Sockel waagerecht oder senkrecht stehen, benutzen Sie bitte eine der beiden Skizzen zum Einbau der Eproms von EXBASIC LEVEL II.



Wenn die beiden Bausteine von EXBASIC LEVEL II in ihren Fassungen stecken, dann schließen Sie den Computer wieder. Drehen Sie aber vorläufig noch nicht die Schrauben wieder ein. Stecken Sie das Netzkabel in die Steckdose und schalten Sie den Computer ein.

So, jetzt wollen wir überprüfen, ob EXBASIC LEVEL II auch richtig eingesteckt ist. Geben Sie dazu den Befehl SYS 37100 ein und drücken Sie die RETURN-Taste. Wenn sich jetzt EXBASIC LEVEL II meldet, dann ist alles in Ordnung. Mit diesem Kommmando können Sie EXBASIC LEVEL II nach jedem Einschalten aktivieren. Mit SYS 37100 wird gleichzeitig der Basic Programmspeicher gelöscht, ein im Speicher befindliches Basicprogramm geht somit verloren (gleiche Wirkung wie NEW). Wir werden später einen Befehl kennenlernen, der uns erlaubt EXBASIC LEVEL II zu reaktivieren, ohne damit ein NEW auszuführen.

Falls sich nach SYS 37100 EXBASIC LEVEL II nicht melden sollte, so ist eines der Eproms nicht richtig eingesteckt worden. Alle Bausteine werden vor Verlassen unseres Hauses mehreren Tests unterzogen. Wir garantieren, daß alle Bausteine in ordnungsgemäßem Zustand sind, wenn sie von uns ausgeliefert werden. Sofern die Eproms, welche Sie erhalten haben, also in ordentlichem mechanischem Zustand sind, kann ein Nicht-Funktionieren praktisch nur am falschen Einsetzen liegen.
Wenn sich EXBASIC LEVEL II nicht ordnungsgemäß meldet, so ziehen Sie noch einmal den Netzstecker(!), öffnen Sie den Computer erneut und überprüfen Sie alles anhand der Skizzen. Falls ein Anschlußpin irgendwo abgeknickt sein sollte oder ähnliches, so ziehen Sie mit äußerster Vorsicht (!!!) den falsch eingesetzten Baustein heraus, biegen den Pin vorsichtig gerade und setzen ihn erneut ein. Passen Sie auf, daß diesmal alles in Ordnung ist.
Überprüfen Sie dann wie beschrieben, ob EXBASIC LEVEL II jetzt ordnungsgemäß arbeitet.

Wenn alles in Ordnung ist und EXBASIC LEVEL II sich nach SYS 37100 (RETURN) auf dem Bildschirm gemeldet hat, dann können Sie den Computer wieder endgültig schließen und auch verschrauben. Ihr Computer ist jetzt zu einer der leistungsfähigsten Maschinen geworden, die derzeit auf dem Microcomputermarkt zu finden sind.

Schauen Sie sich die neuen Befehle gleich einmal an. Geben Sie dazu HELP ein und drücken Sie dann die RETURN-Taste. Jetzt haben Sie die neuen Befehle von EXBASIC LEVEL II vor sich. Nicht aufgelistet werden die Floppy Kurzbefehle (nur bei 2001/3001/4001), die Kassettenrekorderbefehle (nur bei 2001/3001/4001) und die Assembler/Disassemblerbefehle (nur 8001). Das macht aber nichts, wir werden im Laufe der Zeit sowieso alle in EXBASIC LEVEL II vorhandenen Möglichkeiten durchsprechen und kennenlernen.

Wie schon im Vorwort angedeutet, setzt das vorliegende Buch dabei von Ihnen Grundkenntnisse des normalen Commodore Basic voraus. Es ist also kein Lehrbuch für die Programmiersprache Basic im allgemeinen, sondern will Ihnen vielmehr helfen, die unglaubliche Fülle an Möglichkeiten, die Ihnen EXBASIC LEVEL II bietet, aufzuzeigen und zu erklären.

Wenn Sie hingegen unabhängig von EXBASIC LEVEL II Ihren CBM-Computer besser kennenlernen möchten, so weisen wir Sie auf die - genau wie EXBASIC LEVEL II - bei INTERFACE AGE (COMPUTER LIFE) erscheinende CBM/VC-20 Buchreihe hin. Wir können Ihnen die Titel dieser auf Commodore Computer spezialisierten Buchserie deswegen so empfehlen, weil der jüngere der beiden EXBASIC LEVEL II Autoren maßgeblich an der Buchreihe beteiligt ist. Wenn Sie also mit EXBASIC LEVEL II zufrieden sind, dann sollten Sie sich auch einmal die COMPUTER LIFE Bücher ansehen. Wir würden uns auf jeden Fall freuen, Sie in dem einen oder anderen Band wieder zu treffen.

Noch ein Hinweis: EXBASIC LEVEL II ist ein mit größter Sorgfalt geschriebenes und immer wieder getestetes Programm in 6502 Assemblersprache. Jedoch zu behaupten, daß ein Programm dieser Größe und Komplexität fehlerfrei sei, wäre vermessen. Wir können lediglich versichern, daß keine groben Fehler mehr auftreten. Eine Haftung irgendwelcher Art übernehmen wir nicht. Wir hoffen, daß Sie dafür Verständnis haben.

EXBASIC LEVEL II Befehlssyntax

Um Ihnen die Befehle von EXBASIC LEVEL II möglichst einfach und doch umfassend erklären zu können, haben wir uns einiger Konventionen bedient, die Sie sich vorab einprägen sollten, um die Befehlsbeschreibungen besser verstehen zu können. In der Informatik, also der Computerwissenschaft, werden schon seit vielen Jahren solche Konventionen benutzt, um die Befehlssyntax von Programmiersprachen besser beschreiben zu können. Wir haben da also ein bißchen abgeschaut. Aber fangen wir an.

Zunächst einmal ist zu beachten, daß alle Worte aus EXBASIC LEVEL II groß geschrieben sind. Dies bedeutet aber nicht unbedingt, daß sie auch in Großbuchstaben eingegeben werden müssen. Vielmehr müssen sie in jedem Fall ungeshiftet eingegeben werden und sie dürfen auch nicht als Variablennamen benutzt werden.
Wenn Sie eine Graphiktastatur haben (Graphiksymbole auf den Tasten), so werden Sie die EXBASIC LEVEL II Worte normalerweise in Großbuchstaben eintippen (es sei denn, Sie haben auf Groß-/Kleinschreibmodus geschaltet), auf einer Businesstastatur normalerweise in Kleinbuchstaben (es sei denn, Sie haben auf Graphikmodus geschaltet). Wichtig ist in jedem Fall, daß die EXBASIC LEVEL II Befehle ungeshiftet, d.h. ohne die Shifttaste eingegeben werden.

Das, was außer dem Befehlswort noch zu einem Basicbefehl gehört, sind bestimmte Sonderzeichen wie etwa Komma, Doppelpunkt, Klammern u.ä. und die sogenannten Parameter oder Argumente. Nehmen wir ein Beispiel. In dem Befehl 'POKE 59468, 14' ist 'POKE' das Befehlswort, ',' ein Sonderzeichen und 59468 und 14 sind Parameter. In dieser Anleitung zu EXBASIC LEVEL II sind Parameter und Argumente zur Unterscheidung von Befehlsworten klein geschrieben, also 'POKE adresse, wert'. Sonderzeichen stehen so, wie sie zu dem betreffenden Befehl dazugehören.

Um unterscheiden zu können zwischen solchen Parametern, die notwendig sind und solchen, die zwar angegeben werden können, aber nicht müssen, benutzen wir eckige Klammern [] bzw. spitze Klammern ⟨⟩. Argumente und Parameter in eckigen Klammern müssen angegeben werden; wenn sie fehlen, so führt das zu einer Fehlermeldung. Solche in spitzen Klammern können angegeben werden, wenn es sinnvoll erscheint. Werden sie weggelassen, so wird meistens ein bestimmter Ersatzwert angenommen.
Eckige und spitze Klammern dienen lediglich der genannten Unterscheidung, diese Klammern selbst werden nicht eingegeben.
Dazu ein Beispiel: 'FIND [text] ⟨,bereich⟩'. Das Befehlswort lautet FIND (suche nach einem bestimmten Text), es muß ein Text mit angegeben werden und es kann (aber muß nicht) ein Bereich spezifiziert werden, der, wenn er angegeben ist, vom Text mit einem Komma getrennt werden muß. Das klingt vielleicht ein bißchen kompliziert, ist es aber gar nicht. Wir werden bei den einzelnen Befehlen immer wieder Beispiele vorfinden, die zeigen, wie es gemeint ist.

Nun gibt es nicht nur eckige und spitze, sondern auch runde Klammern (). Dazu merken Sie sich am besten folgendes: Ein Ausdruck in runden Klammern muß immer mit angegeben werden, und zwar in runden Klammern. Während also eckige und spitze Klammern lediglich der Unterscheidung 'muß/kann' dienen und selbst nicht mit eingegeben werden, müssen runde Klammern da gesetzt werden, wo sie angegeben sind.
Beispiel: 'SET (spalte,zeile)' bedeutet, daß dem Wort SET eine öffnende Klammer folgen muß, dann eine Spaltenangabe, ein Komma, eine Zeilenangabe und dann eine schließende Klammer, also z.B. SET(12,5).

Sonderzeichen wie Komma, Semikolon etc. müssen - genau wie runde Klammern - mit eingegeben werden, sofern sie nicht in spitzen Klammern stehen. In diesem Fall sind sie nur dann notwendig, wenn auch der gesamte in spitzen Klammern stehende Ausdruck angegeben wird.

Bei den mathematische Funktionen ist als Argument oft ein Großbuchstabe wie X oder Y angegeben. Dies entspricht der normalen mathematischen Schreibweise. Anstelle von X oder Y kann jeder zugelassene numerische Ausdruck stehen, also z.B. auch 5*SIN(12).

Nachfolgend sind einmal ein paar Ihnen sicher bekannte Befehle des normalen Basics in der neuen Syntax geschrieben:

```
LIST <bereich>
PEEK (adresse)
POKE [adresse],[wert]
RND (X)
ABS (X)
PRINT TAB (tabulatorposition);
```

Sie sehen, es ist gar nicht so schwer, wie es vielleicht beim ersten Durchlesen ausssieht. Wenn Sie mit EXBASIC LEVEL II erst ein paar Wochen gearbeitet haben, bereiten Ihnen all die neuen Kommandos sicherlich keine Schwierigkeiten mehr. Und schließlich können Sie sich auch anhand der weiter hinten stehenden fertigen Programme noch die letzten Feinheiten aneignen.

Hilfsfunktionen

Unter Hilfsfunktionen verstehen wir hier EXBASIC LEVEL II Befehle, die im allgemeinen nicht innerhalb eines Programms stehen, sondern dazu dienen, Programme zu schreiben, also Programmierhilfen darstellen. Trotzdem sind alle Hilfsfunktionen auch programmierbar und z.T. ist dies durchaus sinnvoll (z.B bei TRACE, FAST, LETTER, "."). Wir werden bei den einzelnen Befehlen jeweils darauf zu sprechen kommmen.

Am Anfang werden Sie wahrscheinlich gar nicht immer zu den Hilfsfunktionen greifen, selbst wenn dies sinnvoll wäre. So werden Sie z.B. zu Beginn sicher noch öfter LIST eingeben, wo FIND sinnvoller wäre. Gewöhnen Sie sich aber bald die neuen Funktionen an. Dazu mag es am Anfang hin und wieder notwendig sein, daß Sie sich ganz bewußt überlegen, "welcher Befehl kann mir jetzt bei der Lösung dieses Problems helfen". Sie sollten das dann unbedingt tun. Nur so werden Sie mit den vielfältigen Möglichkeiten von EXBASIC LEVEL II voll vertraut.

* FIND [text] (,bereich)

Dieser Befehl ermöglichst die Suche nach einem bestimmten Basicwort, Text oder Variable. Die Zeilen, in denen die Folge gefunden wird, werden aufgelistet. Damit entfällt das oft langwierige mehrmalige Auflisten eines Programms, bis man das Gesuchte gefunden hat.

Beispiele:

```
FIND REM      listet alle Zeilen mit einem REM-Befehl
FIND OPEN     listet alle OPEN-Befehle
FIND "Hallo"  listet alle 'Hallo' in Anführungszeichen
FIND A$       sucht nach der Variablen A$
FIND A$=      sucht nach Wertzuweisungen zu A$
```

FIND sucht nicht nach dem leeren String "" oder einem String, der nur aus Leerzeichen (Spaces) " " besteht. Ebenso kann nicht nach Anführungszeichen '"' oder Kommata ',' gesucht werden.

Zusammen mit FIND kann ein Bereich angegeben werden, innerhalb dessen gesucht werden soll. Dieser Bereich wird wie beim LIST-Befehl spezifiziert, er muß mit einem Komma von dem zu suchenden Basicwort, Text oder Variable getrennt werden.

Beispiele:

```
FIND THEN,1000-5000   sucht zwischen Zeilen 1000 bis 5000
FIND PRINT,-100       sucht vom Anfang bis Zeile 100
FIND A=,100-          sucht ab Zeile 100 bis Programmende
```

FIND ist zwar, wie alle Hilfsfunktionen programmierbar, findet jedoch im Programmablauf keine Übereinstimmungen und bricht den Programmlauf ab.

* AUTO (anfangszeilennummer) (,schrittweite)

Sie wissen, wie lästig es ist, beim Eingeben eines längeren Programmes zu jeder neuen Zeile die Zeilennummer mit einzutippen. Und wenn man aus Versehen einmal die Zeilennummer nicht korrekt geschrieben hat, fügt man die Zeile an einer völlig falschen Stelle hinzu oder überschreibt möglicherweise sogar eine schon vorhandene. All dies erspart Ihnen der Befehl AUTO.
AUTO bewirkt eine automatische Zeilennummernvorgabe, wobei Sie die Anfangszeilennummer und die Schrittweite selbst bestimmen können.

Beispiele:

```
AUTO 100,10    beginnt bei 100 in Zehnerschritten
AUTO 1,1       beginnt bei 1, weiter 2, 3, 4, 5 ...
AUTO           beginnt bei 10 in Zehnerschritten
```

Wie aus dem letzten Beispiel zu ersehen ist, bewirkt AUTO ohne weitere Angaben dasselbe wie AUTO 10,10. Dies gilt aber nur zu Beginn, wenn EXBASIC LEVEL II gerade aktiviert wurde oder wenn vorher NEW eingegeben worden ist.
Andernfalls setzt AUTO ohne weitere Angaben da fort, wo Sie vorher mit AUTO aufgehört haben. Auch die Schrittweite bleibt die gleiche.

Beispiel:

Sie geben AUTO 100,10 ein. Dann weiter...

```
100 REM Dies ist ein Beispiel
110 PRINT "Hallo, wie geht es Ihnen?"
120
```

Der Cursor steht nun hinter Zeile 120. Um den AUTO-Modus zu verlassen, drücken Sie einfach die RETURN-Taste, ohne sonst irgendetwas einzugeben. Der Cursor springt dann zum Zeilenanfang und es erscheint keine neue Zeilennummer mehr. Hoppla.., Sie wollten doch das Programm mit END abschließen. Geben Sie dazu ein...

```
AUTO
```

..und es erscheint...

```
120
```

..und Sie fügen hinzu...

```
120 END
130
```

Drücken Sie RETURN und Sie sind wieder aus dem AUTO-Modus draußen. Ganz einfach, nicht wahr?
Existiert die momentane Zeilennummer bereits, so erscheint ein '*' hinter der letzten Ziffer. Wird eine Zeilennummer nur mit * oder leer mittels RETURN abgeschickt, so wird AUTO abgebrochen und die momentane Zeile nicht gelöscht.
Die höchste Zeilennummer überhaupt ist 63999.

* DEL [bereich]

Dieses Kommando erlaubt ganze Basic Programmbereiche auf einmal zu löschen. Die Syntax ist dieselbe wie bei LIST, allerdings muß zu DEL immer ein Bereich mit angegeben werden. Das verhindert, daß Sie durch Eingabe von DEL und unbeabsichtigtes Drücken der RETURN-Taste das ganze Programm löschen.

Beispiel:

```
DEL 100-400   löscht alle Zeilen zwischen 100 und 400
DEL 5000-     löscht alle Zeilen ab 5000 bis Programmende
DEL -100      löscht alle Zeilen vom Anfang bis 100
```

Seien Sie vorsichtig bei der Verwendung von DEL, damit Sie nicht aus Versehen wichtige Programmteile löschen.

* RENUM (anfangszeilennummer) (,schrittweite)

RENUM renumeriert ein im Speicher befindliches Basicprogramm. Nehmen wir einmal an, folgendes kurze Programm stünde im Speicher:

```
1 REM BEISPIELPROGRAMM
10 PRINT "Dies ist ein ziemlich sinnloses ";
15 PRINT "Programm, das lediglich zur"
50 PRINT "Demonstration dient, weiter nichts!"
```

Geben Sie nun RENUM 100,10 ein und unser kurzes Programm wird zu...

```
100 REM BEISPIELPROGRAMM
110 PRINT "Dies ist ein ziemlich sinnloses ";
120 PRINT "Programm, das lediglich zur"
130 PRINT "Demonstration dient, weiter nichts!"
```

RENUM ohne weitere Zusätze ist gleichbedeutend mit RENUM 10,10, d.h. einer Numerierung in Zehnerabständen, die bei 10 beginnt. Probieren Sie es doch einmal an unserem kurzen Beispielprogramm aus und geben Sie RENUM ein.

Beachten Sie, daß RENUM selbstverständlich alle Zeilennummern hinter den Anweisungen GOTO, GOSUB, RUN, RESTORE, THEN, ELSE, RESUME und LIST entsprechend mitändert. Andernfalls wäre ja das Programm nach der Renumerierung gar nicht mehr lauffähig.

Wird eine Schrittweite von 0 angegeben, so führt dies zu einem ?ILLEGAL QUANTITY ERROR, da nach einem solchen Befehl alle Programmzeilen die gleiche Nummer hätten, mithin das Programm zerstört wäre.
Falls die Umnumerierung eine Bereichsüberschreitung verlangt, d.h. wenn Zeilennummern größer als 63999 kreiert werden müßten, dann wird RENUM gar nicht ausgeführt und es erscheint ebenfalls ein ?ILLEGAL QUANTITY ERROR. Das im Speicher stehende Programm bleibt voll erhalten.

* TRACE / TRACE OFF

Mit TRACE wird der TRACE-Modus in EXBASIC LEVEL II eingeschaltet. Ist TRACE aktiv, so erscheint während der Ausführung von Basic Programmzeilen in den oberen Bildschirmzeilen die momentan ausgeführte Programmzeile und ein reverses Zeichen weist auf den gerade ausgeführten Befehl hin. Dadurch wird es besonders leicht, den Ablauf eines Programms zu verfolgen und evtl. Fehler zu finden.

Wird ein Programm im TRACE-Modus ausgeführt, so läßt sich noch zusätzlich die Geschwindigkeit mit den beiden Shifttasten steuern: Ein Druck auf die linke Shifttaste bewirkt schnellen Ablauf, ein kurzer Druck auf die rechte Shifttaste bewirkt die Ausführung nur eines einzelnen Befehls. Halten Sie die rechte Shifttaste länger niedergedrückt, so wird das Programm mit einer Geschwindigkeit von ca. zwei Befehlen pro Sekunde abgearbeitet.

Ein Programmlauf im TRACE-Modus kann, wie normal, jederzeit durch Drücken der STOP-Taste abgebrochen werden. Wegen der niedrigen Arbeitsgeschwindigkeit des Computers müssen Sie aber die STOP-Taste ein wenig länger gedrückt halten als im Normalmodus. Der TRACE-Modus bleibt solange eingeschaltet, bis TRACE OFF eingegeben wird.

Da der Befehl TRACE auch programmierbar ist, bietet es sich an, ein Programm bis zu dem Bereich, der ausgetestet werden soll, mit normaler Geschwindigkeit laufen zu lassen und dann erst TRACE im Programm einzuschalten. Dann muß nicht das gesamte Programm langsam abgearbeitet werden und trotzdem ist der kritische Bereich, in dem der Fehler zu erwarten ist, leicht zu verfolgen.

* ON / OFF

Beim Aktivieren von EXBASIC LEVEL II mit SYS 37100 wird auch ON eingeschaltet.

Bei den Geräten der Serie 2001/3001 und 4001 wird durch ON die Repeatfunktion (Wiederholfunktion) eingeschaltet. Insbesondere die Cursorsteuerung wird dadurch wesentlich erleichtert. Die Ansprechzeit für Repeat beträgt 1/4 Sekunde, die Wiederholgeschwindigkeit 20 Zeichen pro Sekunde.

Bei allen Serien wird durch ON der Fehlermodus eingeschaltet. Wenn im Fehlermodus ein Programm abläuft und es tritt ein Programmfehler auf, so listet der Computer automatisch die fehlerhafte Zeile auf dem Bildschirm auf und positioniert den Cursor an der Stelle, wo er den Fehler erkannt hat. Dies zeigt zum einen sofort, wo der Fehler liegt und zum anderen ist eine Korrektur mit wenigen Tastendrücken möglich, da der Cursor schon richtig positioniert ist. Natürlich vermag der Computer auch im Fehlermodus lediglich formale Fehler - und nicht etwa solche logischer Art - zu erkennen.

Bei Fehlern in DATA oder INPUT steht der Cursor in jedem Fall direkt hinter der Zeilennummer und nicht beim fehlerhaften Befehl, da er diesen nicht zu erkennen vermag. Wenn Sie die Tasten nicht mit Repeat versehen haben wollen oder keinen Fehlermodus wünschen, so können Sie beides mit OFF abschalten.

* DUMP

Dieses Kommando listet alle benutzten nicht indizierten Variablen wie A...Z, AA...ZZ, A%...Z%, AA%...ZZ%, A$...Z$, AA$...ZZ$. Nicht gelistet werden dagegen indizierte Variablen (Feldvariablen) wie z.B. A$(1).

Die Variablen werden in der gleichen Reihenfolge aufgelistet, in der sie (meist im Programm) definiert wurden. Beachten Sie, daß eine Variable auch dann definiert wird, wenn ihr der Wert 0 zugewiesen wird (z.B. A=0). Eine solche Variable wird folglich auch durch DUMP ausgewiesen.

Beispielprogramm:

```
100 A=5
110 B=20
120 C=A+B
130 END
140 D=B-A
```

Wenn wir nach dem Programmlauf DUMP eingeben, so erscheint:

```
A  = 5
B  = 20
C  = 25
```

Die Variable D wird nicht mit ausgegeben, obwohl sie im Programm steht, da sie beim Programmlauf wegen des END-Befehls in Zeile 130 gar nicht definiert wurde, d.h. ihr kein Wert zugewiesen wurde.

* MATRIX

MATRIX ist der zu DUMP äquivalente Befehl für alle indizierten Variablen (Feldvariablen), also alle Variablen, denen nach dem Namen noch ein Index (.) folgt. Alles über DUMP Gesagte gilt übertragen auch für MATRIX.

Durch die Unterscheidung zwischen DUMP und MATRIX ist es für Sie besonders einfach, einen differenzierten Überblick über die Variablenbelegung Ihrer Programme zu gewinnen. Nehmen wir einmal an, Ihr Programm soll die Matrix A(0)...A(20) von Diskette oder Kassette einlesen. Aus irgendeinem Grund werden aber einige Felder falsch belegt. Sie geben jetzt nur noch MATRIX ein und sehen sofort, in welche Variablen welche Werte eingelesen wurden.

* LETTER / LETTER OFF

Mit LETTER schalten wir den Computer auf Groß-/Kleinschreibmodus um. In diesem Modus sind Kleinbuchstaben ohne Shifttaste und Großbuchstaben zusammen mit Shift auf der Tastatur erreichbar. Dies entspricht praktisch einer normalen Schreibmaschinentastatur. Bei Geräten mit Businesstastatur ist dies nach dem Einschalten des Computers sowieso der Fall.

In den Graphikmodus schalten wir mit LETTER OFF. Dann sind die Großbuchstaben ungeshiftet und diverse Graphikzeichen über Shift erreichbar. Geräte mit Graphiktastatur sind direkt nach dem Einschalten immer in diesem Modus.

LETTER (oder auch LETTER ON) entspricht dem Befehl POKE 59468,14 (Serien 2001/3001/4001) bzw. PRINT CHR$(14) (Serie 8001) , LETTER OFF dagegen POKE 59468,12 bzw. PRINT CHR$(142).

LETTER und LETTER OFF sind Hilfsfunktionen, die zu programmieren durchaus sinnvoll ist. Sie ersparen dann die angeführten POKE- oder PRINT-Befehle.

* FAST / FAST OFF

FAST oder FAST ON schaltet die schnelle Bildschirmausgabe ein. Alle PRINT-Anweisungen und LIST funktionieren jetzt mit weitaus höherer Geschwindigkeit als normal. Dagegen übt FAST auf POKE-Befehle, die den Bildschirm betreffen, keine Wirkung aus, diese laufen also mit gewohnter Geschwindigkeit ab.

Mit FAST OFF wird wieder auf normale Ausgabegeschwindigkeit umgeschaltet. FAST und FAST OFF sind Hilfsfunktionen, die auch in einem Programm sinnvoll sind, damit z.B. eine Tabelle schneller ausgegeben wird, ein Spiel schneller abläuft oder ähnliches.

FAST ist ausschließlich für die Serien 2001 und 3001 gedacht. Bei Geräten der anderen Serien führt FAST in keinem Fall zu einer Erhöhung der Ausgabegeschwindigkeit, da diese Computer grundsätzlich in einem schnellen Ausgabemodus arbeiten. Bei anderen Computern als 2001 oder 3001 mag FAST zu recht merkwürdigen Effekten auf dem Bildschirm führen, sie sollten es in diesem Fall nicht benutzen.

* .

Der Punkt '.' ist in EXBASIC LEVEL II so etwas wie ein Symbol für den Ausdruck "zuletzt abgearbeitete Zeile", z.B. bei der Eingabe, Break etc.

Beispiele:

LIST.   listet die zuletzt eingegebene Zeile

LIST-.	listet bis zur zuletzt eingegebenen Zeile
LIST.-	ab zuletzt eingegebener Zeile bis zum Ende
GOTO.	springt an die zuletzt angesprochene Zeile
RUN.	Programmstart an zuletzt angesprochener Zeile

Da in EXBASIC LEVEL II alle Hilfsfunktionen auch programmierbar sind, ergibt sich für den Punkt in diesem Zusammenhang eine ganz besondere Möglichkeit:

```
100 PRINT "Drücken Sie bitte die Taste 'X'"
110 GET A$ : IF A$<>"X" THEN.
120 PRINT "Sie haben 'X' gedrückt, sehr gut!"
```

Beachten Sie Zeile 110. Auch hier steht der Punkt sozusagen für "zuletzt abgearbeitete Zeile", und das ist logischerweise immer genau die Zeile, in welcher der Punkt selbst steht. In Zeile 110 bewirkt also das 'THEN.' genau dasselbe wie 'THEN 110'. Nur daß der Punkt natürlich Speicherplatz spart. Im Beispiel genau zwei Byte. Außerdem erhöht sich die Übersichtlichkeit des Programms. Hinweis: Der Punkt funktioniert nicht korrekt in der ersten Zeile eines Unterprogrammes, da er dort als Referenz zu der Zeile, in der GOSUB steht, verstanden wird.
Der Punkt als Ersatzsymbol kann bei folgenden Basicworten benutzt werden: LIST, RUN, GOTO, THEN, ELSE.

* MEM

Wie Sie wissen, ist es möglich mit dem Befehl FRE festzustellen, wieviel Speicherplatz noch frei ist und somit für Programme oder Variablen verbleibt. FRE muß ein Argument mitgegeben werden, also z.B. PRINT FRE(0). Die Zahl in Klammern hat aber sonst weiter keine Bedeutung.

MEM gibt nun eine differenzierte Auskunft darüber, wie der Speicherplatz des Computers belegt ist, und zwar wie folgt:

MAIN MEMORY:	BYTES	= Gesamtkapazität
PROGRAM:	BYTES	= Programm
VARIABLES:	BYTES	= normale Variablen
ARRAYS:	BYTES	= Feldvariablen
STRINGS:	BYTES	= Stringvariablen
REK:	BYTES	= REK Speicherplatz
FREE:	BYTES	= noch freier Platz

MEM benötigt im Gegensatz zu FRE kein Argument, Sie geben einfach MEM (RETURN) ein.

Um die Angaben von MEM auf einem Drucker auszugeben, tippen Sie folgendes ein:

```
OPEN 4,4 : CMD 4 : MEM
```

..und drücken dann die RETURN-Taste. Vergessen Sie nicht, das geöffnete File wieder zu schließen mit...

```
PRINT§4 : CLOSE 4
```

* HIMEM [adresse]

Mit HIMEM können Sie den Basic-Speicherbereich nach oben begrenzen. Für Basic steht danach nur noch der Bereich von 1024 bis zur angegebenen Endadresse zur Verfügung, der Rest kann für Maschinenprogramme oder spezielle Daten genutzt werden.

HIMEM ist eigentlich nur für den fortgeschrittenen Programmierer interessant, der seinen Computer besser kennt und nicht ausschließlich in Basic programmiert. Insbesondere zusammen mit dem eingebauten Terminal Interface Monitor TIM, der über das EXBASIC LEVEL II Kommando GO erreicht werden kann, und dem integrierten Assembler/Disassembler (nur Serie 8001) erleichtert HIMEM das Programmieren in Maschinensprache und Assembler.

Beachten Sie, daß HIMEM die Befehle CLR und REK OFF impliziert. Die zusammen mit HIMEM angegebene Adresse kann jeden Wert zwischen 1260 und 32768 haben.

* GO

Dieser Befehl stellt eine Verbindung zwischen Basic bzw. EXBASIC LEVEL II und Maschinensprache dar. Sie wissen sicherlich, daß Ihr Computer als Herzstück einen Mikroprozessor MP 6502 hat. Dieser Mikroprozessor alleine vermag nun kein Basic und erst recht kein EXBASIC LEVEL II zu verstehen, vielmehr kann er nur mit Zahlen operieren. Diese Zahlen, sofern sie für die MPU (Micro Processor Unit) 6502 etwas bedeuten, nennen wir Maschinensprache. Um sie einzugeben, bedarf es eines speziellen Monitors, wir benutzen dazu den bereits erwähnten Terminal Interface Monitor TIM. Und in diesen gelangen wir, wenn wir GO eingeben und dann RETURN drücken (Serie 8001: Einsprung in EXMON).

Auf dem Bildschirm erscheint nach GO:

```
C*   PC   IRQ  SR AC XR YR SP
.;  986E A4DC 00 00 30 30 FA
```

..oder ähnliches. Dabei bedeuten:

PC	= program counter	= Programmzähler
IRQ	= interrupt request	= IRQ-Vektor
SR	= status register	= Statusregister
AC	= accumulator	= Akkumulator
XR	= X register	= Indexregister X
YR	= Y register	= Indexregister Y
SP	= stack pointer	= Stackzeiger

Die Zahlen unterhalb der Bezeichnungen stellen die Inhalte der entsprechenden Zähler, Register und Zeiger dar.
Dies ist aber wirklich nur noch etwas für Maschinenprogrammierer. Wenn Sie sich (noch) nicht dazu zählen, dann geben Sie jetzt am besten X (RETURN) ein und Sie befinden sich wieder im EXBASIC LEVEL II.

Wenn TIM aufgerufen ist, stehen Ihnen zusätzlich folgende Befehle zur Verfügung (Serie 8001: siehe EXMON):

G = Ausführung eines Maschinenprogramms
L = Laden eines Maschinenprogramms
M = Anzeige eines Speicherbereichs
R = Anzeige der Registerinhalte
S = Saven eines Maschinenprogramms
X = Rücksprung zu EXBASIC LEVEL II

TIM arbeitet prinzipiell hexadezimal. Mit den EXBASIC LEVEL II Befehlen HEX$ und DEC sind aber jederzeit Umrechnungen 'dezimal - hexadezimal' in beide Richtungen möglich.

Beispiele:

```
G 0000                     Programmausführung mit USR-Vektor
L "NAME",08                lädt Programm NAME von Diskette
M 033A 0340                listet Teil des 2. Kassettenpuffers
R                          zeigt Registerinhalte an
S "NAME",01,033A,0340 saved auf Kassette ab (08= Floppy)
X                          Rücksprung zu EXBASIC LEVEL II
```

Beachten Sie zu S, daß die hintere Adresse (0340) die Endadresse des Maschinenprogramms plus eins ist, das letzte Datenbyte befindet sich also in 033F.

Die Eingabe von Maschinenprogrammen, die in Hexcode vorliegen, ist mit dem Kommando M möglich. Zuerst wird der gewünschte Speicherbereich mit M aufgelistet und sodann die alten Adressinhalte durch das Maschinenprogramm überschrieben. Wichtig ist, daß dabei das gelistet Format genau eingehalten wird. Auch hier ist es notwendig, geänderte Zeilen mittels der RETURN-Taste zu quittieren.
Wenn die Bildschirmlänge nicht ausreicht, um das Maschinenprogramm auf einmal einzugeben, so muß es in mehreren kleineren Einheiten eingetippt werden. Mit M wird dazu der jeweils weitere Speicherplatz aufgelistet.
Ein auf diese Art und Weise eingegebenes Programm kann nun mittels S auf Kassette oder Diskette abgespeichert werden. Es ist später mit einem normalen LOAD-Befehl wieder einlesbar.

Bei Geräten der Serien 2001 und 3001 steht für Maschinenprogramme der zweite Kassettenpuffer von 826 bis 1017 (dezimal) zur Verfügung. Sofern ihr Programm da hineinpaßt, sollten Sie es auch dort unterbringen. Es ist dann nämlich absolut sicher, auch durch NEW wird es nicht gelöscht.
Längere Maschinenprogramme werden am besten im oberen Basicspeicherbereich bis 32768 (dort beginnt das Bildschirm RAM) abgelegt. Dies gilt prinzipiell bei Geräten der Serien 4001 und 8001. Diese benutzen den zweiten Kassettenpuffer im Zusammenhang mit Floppy-Kommandos und der TAB-Taste. Zur Speicherbegrenzung steht HIMEM zur Verfügung.

Beispiel (für 32k RAM):

HIMEM 30768 schafft 2000 Byte Speicherplatz.

SPACE / SPACE OFF

Dies ist eine Option in EXBASIC LEVEL II, auf die Sie schon bald nicht mehr verzichten werden wollen. Durch die Eingabe von SPACE oder SPACE ON wird auf formatiertes Listen geschaltet. Bitte verwechseln Sie dies nicht mit den Befehlen für formatiertes Drucken PRINT USING und PRINT USING#.

Im SPACE-Modus werden beim Auflisten eines Programmes mittels LIST vor und hinter den einzelnen Basickommandos Leerzeichen (Spaces) eingefügt. Dadurch wird ein wesentlich übersichtlicheres Lesen des Programmes möglich als das normalerweise der Fall ist. Der SPACE-Modus wirkt auch bei Listings auf einen Drucker. Dies ist eine besonders angenehme Option von EXBASIC LEVEL II.

Die im SPACE-Modus eingefügten Leerzeichen belegen keinen Speicherplatz, sie werden also nicht wirklich in das Programm eingefügt, sondern erscheinen nur beim LIST-Vorgang.
Auch wenn eine im SPACE-Modus aufgelistete Zeile geändert und danach dem Computer mit RETURN übergeben wird, so werden diese Leerzeichen ignoriert. Sie können also auch im SPACE-Modus nach Herzenslust im Listing korrigieren und ändern.
Übernommen werden lediglich Leerzeichen zwischen Anführungszeichen und in REM- und DATA-Zeilen.

Bitte beachten Sie, daß der SPACE-Modus nur auf LIST wirkt, hingegen nicht auf FIND, TRACE oder den Fehlermodus.

SPACE OFF schaltet um auf normales Listen. Dies entspricht der üblichen Syntax in EXBASIC LEVEL II.

Unabhängig davon, ob der SPACE-Modus eingeschaltet ist oder nicht, können Sie in EXBASIC LEVEL II ein gerade laufendes Listing auf dem Bildschirm jederzeit einfrieren durch Drücken der Taste '@' (2001/3001/4001) bzw. ':' (8001). Eine Fortführung des Listings ist durch Drücken einer beliebigen Taste (2001/3001/4001) bzw. durch Drücken der Taste 'Pfeil nach links' (8001) möglich.
Um den Listvorgang ganz abzubrechen, bedienen wir uns, wie üblich, der STOP-Taste.

Hinweis: Normalerweise ignoriert EXBASIC LEVEL II Leerzeichen in Programmzeilen. Deshalb ist es auch möglich, im SPACE-Modus gelistete Programme mit RETURN zu übernehmen, ohne daß dadurch die gelisteten Leerzeichen ins Programm übernommen werden. In REM- und DATA-Zeilen sowie zwischen Anführungszeichen werden selbstverständlich auch in EXBASIC LEVEL II Leerstellen akzeptiert.

Falls Sie jedoch grundsätzlich Leerzeichen in Ihren Programmen zulassen wollen, so können Sie EXBASIC LEVEL II darauhin wie folgt umschalten:

```
POKE 124, PEEK (124) OR 64
```

Programmzeilen mit Leerzeichen sind in EXBASIC LEVEL II zwar eigentlich gar nicht nötig, da ein formatiertes Listen jederzeit im SPACE-Modus möglich ist, aber wenn Sie es mögen, schalten Sie um. Und wenn es Ihnen nicht mehr gefällt, stellen Sie mit...

```
POKE 124, PEEK (124) AND 191
```

..wieder den Ursprungszustand her.
Wenn Sie in Speicherzelle 124 einen falschen Wert eingeben, kann es sein, daß Sie versehentlich TRACE, SPACE oder ON ERROR GOTO ein- oder ausschalten.

* STOP ON / STOP OFF

Wenn Sie schon einmal mit Maschinenprogrammierung zu tun hatten, dann wissen Sie auch, wie unangenehm es ist, wenn ein solches Programm noch einen Fehler aufweist und trotzdem gestartet wird. In den allermeisten Fällen "verabschiedet" sich der Computer und ist nur durch Aus- und Wiedereinschalten funktionstüchtig zu machen. Nur - das Programm ist dann natürlich verloren.

Auch hier hilft Ihnen EXBASIC LEVEL II. Durch den Befehl STOP ON wird der Computer in den STOP-Modus geschaltet. In diesem Modus können Sie jedes Maschinenprogramm durch Drücken der STOP-Taste unterbrechen. Das funktioniert auch dann noch, wenn sich das Maschinenprogramm in einer Endlosschleife befindet, einen ungültigen OP-Code enthält und was sonst noch so alles passieren kann.

Im Gegensatz zur allgemeinen EXBASIC LEVEL II Befehlssyntax darf bei STOP ON das Wort ON nicht weggelassen werden. STOP ohne ON erfüllt nämlich die altbekannte STOP-Funktion. STOP OFF schaltet zurück auf den normalen Modus.

Der STOP-Modus wird unwirksam, wenn Ihr Maschinenprogramm den Befehl SEI (set interrupt disable) enthält oder den IRQ-Vektor verändert. Ein vernünftiger Rücksprung ist natürlich auch dann nicht mehr möglich, wenn die Zero-Page in wesentlichen Bestandteilen zerstört ist.

Den Fachleuten sei gesagt: STOP ON setzt die Abfrage auf die STOP-Taste grundsätzlich vor jeden anderen Interrupt-Sprung.

* HELP

Dies zeigt die EXBASIC LEVEL II Kommandos auf dem Bildschirm an. So können Sie sich also, gerade zu Beginn, jederzeit einen Überblick über all die neuen Befehle verschaffen.
Nicht aufgelistet werden die Floppy-Kurzbefehle, die Assemblerbefehle und viele EXBASIC LEVEL II Features, die sich aus Kombinationen ergeben.

HELP funktioniert auch zusammen mit einem Drucker. Geben Sie dazu...

OPEN 4,4 : CMD 4 : HELP

..ein und drücken dann die RETURN-Taste. Vergessen Sie nicht, das geöffnete File wieder mit...

PRINT#4 : CLOSE4

..zu schließen. Andernfalls werden auch weitere Kommandos auf den Drucker ausgegeben.

* HELP*

Ähnlich wie HELP wird der Befehlsvorrat des normalen Basics ausgegeben. Auch HELP* funktioniert zusammen mit einem Drucker.

Mit HELP und HELP* haben Sie also die Möglichkeit, sich sämtliche Befehle Ihres Computers auflisten zu lassen. So behalten Sie immer einen Überblick über das, was Ihr Gerät eigentlich kann.

* BASIC

Dies ist eigentlich ein ziemlich sinnloses Kommando, es bewirkt nämlich eine Rückkehr von EXBASIC LEVEL II zum normalen Commodore Basic. Und dazu besteht ja nun wirklich kein Grund. Zu dieser Überzeugung sind Sie im bisherigen Verlauf dieses Handbuchs sicherlich auch gelangt.

Einziges überhaupt mögliches Argument: In EXBASIC LEVEL II laufen die Programme ca..7 bis 8 % langsamer. Normalerweise spielt das aber gar keine Rolle und bei besonders zeitkritischen Aufgaben wie etwa Maschinensteuerungen ist die Programmierung in Assembler sowieso unerläßlich. Außerdem sparen viele EXBASIC LEVEL II Kommandos durch ihre Mächtigkeit auch viel an Zeit wieder ein. So erspart z.B. DISPOSE komplizierte, zeitaufwendige Techniken beim Aussprung aus Schleifen oder Unterprogrammen.

Die Rückkehr zu EXBASIC LEVEL II ist jederzeit durch PRINT USR(0) möglich. Ein augenblicklich gespeichertes Programm geht dabei nicht verloren.

Sie sollten an einem EXBASIC LEVEL II Programm im normalen Basic keine Veränderungen vornehmen, da EXBASIC LEVEL II Kommandos vom normalen Basic logischerweise nicht verstanden werden. Vielmehr werden sie in andere Befehle des normalen Basics umgewandelt, was natürlich zu Unsinn führt. Wenn Sie mit PRINT USR(0) wieder zu EXBASIC LEVEL II zurückkehren, sieht das Programm wieder ganz normal aus.

* MERGE ("name") / MERGE* ["name"]

MERGE erlaubt Ihnen, zwei Programme zusammenzufügen, die beliebige, aber unterschiedliche Zeilennummern haben. Damit können Sie z.B. verschiedene Programmteile zu einem Ganzen zusammenfassen.

MERGE ist eigentlich ein FAST TAPE Befehl für die Serien 2001/3001 und 4001. Bei diesen Serien funktioniert es zusammen mit dem Kassettenrekorder. Um sich hierüber näher zu informieren, schlagen Sie bitte im Kapitel 'FAST TAPE' nach.

Zusammen mit einer Floppydisk funktioniert MERGE, wenn Sie hinter den Namen den Zusatz '*' anhängen. MERGE* von Floppy ist in EXBASIC LEVEL II für alle Serien implementiert, insbesondere auch für die Serie 8001. Bei MERGE* muß - im Unterschied zu MERGE - ein Programmname in Anführungszeichen angegeben werden.

MERGE/MERGE* lädt das durch den Namen spezifizierte Programm, ohne dadurch das gerade im Speicher stehende Programm zu zerstören. Auf diese Weise werden also zwei Programme oder Programmteile zusammen- oder ggf. ineinandergefügt.

Die zwei Programme sollten verschiedene Zeilennummern haben. Falls das nicht gegeben ist, gehen Sie am besten wie folgt vor. Laden Sie das Programm, das das andere teilweise ersetzen soll, als erstes in den Programmspeicher (mit normalem LOAD). Laden Sie sodann das zweite, dessen Zeilennummern Sie teilweise ersetzen wollen, mit MERGE bzw. MERGE* hinzu. In den überlappenden Programmbereichen finden Sie nun doppelte Zeilennummern vor. Wenn Sie jetzt aber diese doppelten Zeilennummern jeweils einmal und einzeln (also nicht mit DEL) löschen, so werden Sie feststellen, daß immer nur die erste Zeile gelöscht wird und die zweite alleine übrigbleibt. Und genau das wollten wir ja erreichen.

Dieser Trick ist natürlich nur im Direktmodus möglich, nicht dagegen, wenn MERGE/MERGE* programmiert wird.

Beispiel zu MERGE:

Folgendes Programm steht im Programmspeicher...

```
100 PRINT "Programm 1 Anfang"
400 PRINT "Ende von Programm 1"
```

Sie geben nun...

```
MERGE* "PROGRAMM 2"
```

..ein um Programm "PROGRAMM 2" von Diskette nachzuladen. Das Gesamtprogramm sieht dann so aus...

```
100 PRINT "Programm 1 Anfang"
200 PRINT "Hier ist Programm 2"
210 PRINT "Ende von Programm 2"
400 PRINT "Ende von Programm 1"
```

Graphikbefehle

Die Graphikbefehle in EXBASIC LEVEL II erlauben es, in besonders einfacher Weise Graphiken, Plottings, Bargraphiken u.ä. zu erstellen. Wenn Sie ein Gerät der Serie 8001 besitzen, so haben Sie sogar die doppelte Bildschirmfläche zur Verfügung.

* PRINT@ [position],

PRINT@ (PRINT AT) setzt den Cursor an die mit Position angegebene Stelle des Bildschirms. Damit ist der gesamte Bildschirm auf einfache Weise mit einem einzigen Befehl erreichbar.

Bei den Geräten mit 40 Zeichen Bildschirmbreite (2001/3001/4001) kann als Position jede Zahl zwischen 0 und 999 angegeben werden, bei der Serie 8001 mit 80 Zeichen Bildschirmbreite jede Zahl zwischen 0 und 1999.
0 kennzeichnet dabei die Position links oben (HOME Position des Cursors), 999 bzw. 1999 die Position rechts unten.

Beispiele:

```
PRINT@ 497,"Hallo" druckt "Hallo" in die Mitte (40 Z.)
PRINT@ 997,"Hallo" druckt in die Mitte bei 80 Zeichen
```

Um bestimmte Zeilen und Tabulatorstellen mit PRINT@ anzufahren, können Sie folgende Formeln benutzen:

```
für 2001/3001/4001: PRINT@ Z*40+S
für 8001:           PRINT@ Z*80+S
```

Dabei steht die Variable Z für Zeile, die Variable S für Spalte. Nehmen wir also einmal an, Sie möchten den Ausdruck "EXBASIC LEVEL II" an die 20. Position der 7. Zeile drucken, dann schreiben Sie einfach...

```
für 2001/3001/4001: PRINT@ 7*40+20,"EXBASIC LEVEL II"
für 8001:           PRINT@ 7*80+20,"EXBASIC LEVEL II"
```

..oder noch einfacher...

```
für 2001/3001/4001: PRINT@ 300,"EXBASIC LEVEL II"
für 8001:           PRINT@ 580,"EXBASIC LEVEL II"
```

Das Komma hinter PRINT@ kann entfallen, wenn eine Variable, eine Funktion oder ein String folgen, so daß die Eindeutigkeit gewahrt bleibt.

Beispiele:

```
PRINT@ 100 "EXBASIC LEVEL II"  ..ist eindeutig
PRINT@ 100 45  ..falsch (Space ist kein Trennungszeichen)
PRINT@ 100,45  ..so ist es korrekt
```

* HPLOT [numerischer ausdruck]

Das Wort HPLOT steht für 'horizontales Plotten'. Und 'Plotten' ist im Grunde nur ein anderes Wort für 'Zeichnen'. HPLOT erlaubt somit das Plotten in horizontaler Richtung. Dies ist besonders dazu geeignet, waagerechte Balkengraphiken (Bargraphs) zu erstellen.

Die Auflösung von HPLOT beträgt 320 (40 Zeichen Bildschirm) bzw. 640 (80 Zeichen Bildschirm, CBM 8001) Punkte in der Horizontalen und 25 Punkte vertikal.
Das bedeutet, daß der numerische Ausdruck zu HPLOT jeden Wert zwischen 0 und 319 bzw. 639 annehmen darf. Übrigens, unter 'numerischer Ausdruck' ist einfach ein Term zu verstehen, dessen Ergebnis eine Zahl (Nummer), und nicht z.B. ein String ist.
Beispiele für numerische Ausdrücke sind...

```
5
78*12+14
COS(1.5)
ABS(INT(TAN(0.5)*45)-2)/9
SIN(X)
```

Auch Letzteres ist ein numerischer Ausdruck, da sich für jeden Wert X ein genaues Ergebnis bestimmen läßt.

Der Befehl HPLOT veranlaßt nun einen Plot ab der augenblicklichen Cursorposition nach rechts um soviel Punkte, wie im numerischen Ausdruck angegeben sind. Danach steht der Cursor eine Zeile tiefer exakt an der Tabulatorposition, von der vorher losgeplottet wurde. Das klingt recht kompliziert, wir wollen uns aber gleich einmal ein Beispiel dazu anschauen.

Beispiel:

```
10 PRINT TAB(4);
20 HPLOT (SIN(X)+1)*120 : X=X+.1 : GOTO.
```

Dieses Beispiel druckt fortlaufend eine Sinuskurve aus, die ihren tiefsten Punkt bei Tabulatorposition 4 hat. Der einmal gesetzte Tabulator gilt, wie beschrieben, für alle folgenden HPLOT-Befehle. Deswegen ist auch in Zeile 20 kein PRINT TAB(4) mehr nötig.
Beachten Sie, daß zu SIN(X) jeweils noch eine 1 addiert wird. Das ist notwendig, weil HPLOT nur einen positiven Parameter kennt, die Sinusfunktion aber auch negative Werte ausgibt. Die Multiplikation mit 120 nun zieht sozusagen die entstehende Kurve auseinander, so daß sie den Bildschirm füllt. Bei einem CBM 8001 können Sie die 120 ruhig einmal durch einen größeren Wert ersetzen.

Erinnern Sie sich noch an die Bedeutung des Punktes '.' in 'GOTO.' in Zeile 20? Wenn nicht, schlagen Sie noch einmal bei 'Hilfsfunktionen 6' nach! Sie sehen, wie sich die vielfältigen Möglichkeiten von EXBASIC LEVEL II immer wieder miteinander kombinieren lassen.

* VPLOT [numerischer ausdruck]

VPLOT bedeutet 'vertikales Plotten' und ist ein HPLOT durchaus nicht unähnlicher Befehl. Während aber HPLOT von links nach rechts zeichnet, tut VPLOT das gleiche von unten nach oben.

VPLOT bewirkt einen Plot ab der augenblicklichen Cursorposition um soviel Punkte nach oben, wie im numerischen Ausdruck angegeben sind. Danach steht der Cursor eine Tabulatorposition weiter rechts. Lediglich bei einem Plot in der allerletzten Bildschirmspalte entfällt diese automatische Tabulierung, dort wäre sie ja auch nicht sehr sinnvoll.

Die Auflösung bei VPLOT beträgt 200 Punkte in der Vertikalen und 40 (40 Zeichen Bildschirm) bzw. 80 (80 Zeichen Bildschirm) Punkte in der Horizontalen.
Der zu VPLOT gehörige numerische Ausdruck darf folglich Werte zwischen 0 und 199 annehmen. Werte bis 255 werden in den Maximalwert 199 umgewandelt, bei noch größeren Werten erfolgt eine Fehlermeldung. Negative Werte sind auch bei VPLOT, genau wie bei HPLOT, nicht zulässig.

Beispiel zu VPLOT:

```
10 SC=999 : LI=39 :REM Serien 2001/3001/4001
10 SC=1999 : LI=79 :REM Serie 8001

20 FOR X = SC-LI TO SC : PRINT@ X;
30 FOR I = 0 TO SC-X : VPLOT I : NEXT I
40 NEXT X
```

Tippen Sie das Beispiel einmal ein und probieren Sie es aus. Das ist doch ganz lustig, nicht wahr?

Beachten Sie, daß nach VPLOT kein Semikolon oder Komma folgt. Trotzdem rutscht der Cursor nach VPLOT nicht eine Zeile tiefer (wie es etwa bei einem PRINT-Befehl passiert), sondern bleibt in derselben Zeile und rückt nur um eine Tabulatorposition weiter nach rechts (Ausnahme letzte Spalte siehe oben).

Sowohl HPLOT als auch VPLOT löschen die Zeile bzw. Spalte, in welche sie plotten, nicht vorher, sondern überschreiben sie lediglich so weit, wie der Plot eben reicht. Umrahmungen, Überschriften, Kennzahlen u.ä. werden also von den Plotbefehlen nicht beeinflußt.

Beispiel:

```
10 S= .16 : P= 960 :REM 2001/3001/4001
10 S= .08 : P= 1920 :REM 8001

20 PRINT@ P;
30 FOR X= 0 TO 6.28 STEP S
40 VPLOT (SIN(X)+1)*100
50 NEXT X
60 PRINT@0
```

* SET (spalte,zeile)

Der Befehl SET und die beiden folgenden, dazugehörigen Kommandos RESET und POINT beziehen sich auf eine Graphikaufteilung des Bildschirms in 4000 (Serien 2001, 3001, 4001) bzw. 8000 (Serie 8001) Graphikpunkte. Die Auflösung beträgt 50 (vertikal) mal 80 bzw. 160 (horizontal) Punkte. Damit können Sie schon durchaus ansehnliche Graphiken erstellen.

SET dient dazu, einen Graphikpunkt zu setzen. Dabei spricht das Argument (0,0) den Punkt links oben (HOME Position) und (79,49) bzw. (159,49) den Punkt rechts unten an. Zu große Werte, die aber noch kleiner als 256 sind, werden in die jeweiligen Maximalwerte umgewandelt. Argumente größer als 255 erzeugen einen ?ILLEGAL QUANTITY ERROR.

Beispiele:

```
SET (12,45)   setzt den Graphikpunkt (12,45)
SET (255,255) setzt auf die äußerste Position
SET (0,0)     setzt einen Punkt links oben
```

* RESET (spalte,zeile)

Während SET den angesprochenen Graphikpunkt setzt, wird dieser duch RESET gelöscht (zurückgesetzt).

Beispiele:

```
SET (5,7)   setzt Punkt (5,7)
RESET (5,7) löscht Punkt (5,7) wieder
```

Ein, allerdings ziemlich unsinniges, kurzes Beispielprogramm soll Ihnen einmal zeigen, wie SET und RESET funktionieren. Beachten Sie dabei, daß RND(X) in EXBASIC LEVEL II für X größer als 1 ganzzahlige Werte zwischen 1 und X erzeugt.

Beispiel:

```
100 GR= 79  :REM Serien 2001/3001/4001
100 GR= 159 :REM Serie 8001

110 SET (RND(GR),RND(49))
120 RESET (RND(GR),RND(49))
130 GOTO 110
```

In Zeile 110 wird ein Punkt gesetzt, der durch RND zufällig bestimmt wird, in Zeile 120 wird ein anderer, ebenfalls zufällig ausgerechneter Punkt gelöscht. Somit baut sich mit der Zeit ein sich ständig veränderndes Muster auf dem Bilschirm auf, ohne daß dieser einmal ganz voll wäre. Natürlich gibt es für die EXBASIC LEVEL II Graphikbefehle wesentlich nützlichere Anwendungen. Eine Auswahl davon finden Sie in den Beispielproammen am Ende dieser Anleitung.

* POINT (spalte,zeile)

Während SET und RESET zum Setzen bzw. Rücksetzen eines Graphikpunktes benutzt werden, können Sie mit POINT abfragen, ob ein bestimmter Punkt gesetzt oder nicht gesetzt ist. Damit erhalten Sie also eine vollständige Kontrolle über die Graphikaufteilung des Bildschirms.

POINT ist eine sogenannte 'Boole'sche Funktion', d.h. das Ergebnis kann nur entweder 0 oder -1 sein. Wenn es 0 ist, so war der abgefragte Punkt nicht gesetzt, bei -1 dagegen war er gesetzt.

Beispiel:

```
10 INPUT "Spalte,Zeile"; S,Z
20 IF POINT (S,Z) PRINT "set" ELSE PRINT "reset"
30 GOTO 10
```

Dieses kurze Programm erlaubt es, beliebige Graphikpunkte daraufhin zu testen, ob sie gesetzt oder nicht gesetzt sind. In Zeile 10 können Sie einen bestimmten Punkt eingeben, der dann in Zeile 20 abgefragt wird. Beachten Sie, daß ELSE ein neuer EXBASIC LEVEL II Befehl ist, der eine Alternative zu THEN darstellt. Mit anderen Worten, wenn THEN nicht ausgeführt wird (weil der Punkt nicht gesetzt und somit die Bedingung nicht erfüllt ist), dann fährt das Programm bei ELSE fort.

Hinweis: Die Graphikbefehle SET, RESET und POINT können prinzipiell auch ein Argument ohne Klammern mitbekommen:

Beispiel:

```
SET 12,3   bewirkt genau dasselbe wie...
SET (12,3) und spart sogar zwei Byte Speicherplatz.
```

Wir empfehlen Ihnen aber, die Klammern jeweils mit einzugeben. Dies erhöht nämlich ganz beträchtlich die Übersichtlichkeit, wohingegen die paar Bytes, die mehr verbraucht werden, normalerweise keine Rolle spielen. Und wenn Sie schon ein bißchen länger beim Programmieren dabei sind, dann wissen Sie sicher auch, daß die Übersichtlichkeit eines Programmes eine ganze Menge Wert ist, oder?!

Immerhin, wenn der Speicherplatz einmal wirklich knapp werden sollte, so können Sie durch das Weglassen der Klammern ein paar Bytes einsparen, genauer gesagt, pro Klammer ein Byte.



Mathematische Funktionen

Die in EXBASIC LEVEL II vorhandenen mathematischen Funktionen sind durchweg solche, die von allgemeinem Interesse sein dürften und nicht nur für mathematische Formeln verwendet werden können.

Nicht enthalten sind dagegen Funktionen, die nur für ganz spezielle Anwendungen interessant sind, wie etwa Logarithmusfunktionen, hyperbolische Funktionen, Matrixbefehle oder ähnliches. Es gibt mittlerweile eine ganze Reihe von Literatur, in der beschrieben wird, wie solche speziellen Funktionen mit Hilfe der vorhandenen Möglichkeiten nachgebildet werden können. Im Bedarfsfall sollten Sie dort nachschlagen.

So, genug der Vorrede, beschäftigen wir uns mit den mathematischen Funktionen, die EXBASIC LEVEL II bietet.

* MAX (variablenliste)

Dieser Befehl sucht in Bruchteilen von Sekunden aus einer Liste von Variablen den größten Wert heraus. Die einzelnen Variablen oder Zahlen müssen durch Kommata voneinander getrennt sein.

Beispiel:

```
A= MAX (12,-5,78,3,55,.321,9)
```

Geben Sie jetzt...

```
PRINT A
```

..ein, so erhalten Sie als Ergebnis...

```
78
```

Probieren wir noch ein bißchen weiter mit...

```
PRINT MAX (A,99)
```

.. liefert den Ausdruck...

```
99
```

Das ist eigentlich ganz einfach, nicht wahr?

* MIN (variablenliste)

Für MIN gilt alles schon zu MAX Gesagte. Einziger Unterschied ist, daß MIN im Gegensatz zu MAX nicht den größten, sondern den kleinsten Wert aus der Variablenliste heraussucht.

* FRAC (X)

Sicherlich bekannt ist Ihnen der Basicbefehl INT. Während nun INT den Nachkommateil einer Zahl abschneidet und nur den Vorkommateil zurückläßt, wirkt FRAC genau umgekehrt. FRAC schneidet einer Zahl den Vorkommateil ab, der Dezimalteil bleibt übrig. Programmierbare Taschenrechner weisen häufig eine FRAC-Funktion auf.

Beispiele:

```
INT (12.75)  liefert den Wert 12
FRAC (12.75) liefert den Wert .75
FRAC (-8.5)  liefert den Wert -.5
FRAC (.2)    liefert den Wert .2
```

Wie das dritte Beispiel zeigt, wird durch FRAC nicht das Vorzeichen geändert.

* ROUND (X,Y)

ROUND (X,Y) rundet die Zahl X auf soviel Nachkommastellen, wie in Y angegeben sind.

Beispiele:

```
ROUND (12.527854,2) ergibt den Wert 12.53
ROUND (12.523854,2) ergibt den Wert 12.52
ROUND (12.523854,0) ergibt den Wert 13
ROUND (-2.527854,6) ergibt den Wert -2.527854
ROUND (-2.527854,7) ergibt den Wert -2.527854
ROUND (-5.8,0)      ergibt den Wert -6
```

Statt ROUND (X,0) wie im letzten Beispiel können wir auch einfach ROUND (X) schreiben. ROUND (X) wirkt also ähnlich wie INT, aber die Nachkommastellen werden nicht abgeschnitten, sondern gerundet.

Beispiel:

```
INT (12.5)   ergibt den Wert 12
ROUND (12.5) ergibt den Wert 13
```

Wenn in dem Ausdruck ROUND (X,Y) die Zahl in Y nicht ganzzahlig sein sollte, so wird vorher ihr Nachkommateil abgeschnitten (entspricht INT).

Beispiel:

```
ROUND (12.54321,2.8) ist dasselbe wie...
ROUND (12.54321,2)   und führt zu dem Wert 12.54
```

Sie sehen, daß aus der 2.8 eine 2 wird, daß also nicht gerundet wird (was ja 3 ergäbe), sondern geschnitten. Soll auch hier gerundet werden, so schreiben Sie am besten...

```
ROUND (X, ROUND (Y))
```



* ODD (X)

ODD prüft eine Zahl daraufhin, ob sie gerade oder ungerade ist. Bei geradem X liefert ODD (X) den Wert 0, bei ungeradem X dagegen -1. ODD ist also eine 'Boole'sche Funktion'.

Beispiele:

```
ODD (5)  liefert den Wert -1 (also ungerade)
ODD (88) liefert den Wert 0 (also gerade)
ODD (-7) liefert den Wert -1
ODD (-8) liefert den Wert 0
```

Wenn die Zahl X nicht ganzzahlig ist, so führt ODD vor der Prüfung auf gerade oder ungerade INT aus.

Beispiel:

```
ODD (2.5) ist dasselbe wie...
ODD (2)   und liefert den Wert 0 (also gerade)
```

Zusammen mit dem EXBASIC LEVEL II Befehl ELSE läßt sich eine Abfrage auf gerade/ungerade besonders einfach programmieren.

Beispiel:

```
10 INPUT "Zahl"; Z
20 IF ODD (Z) THEN PRINT "ungerade" ELSE PRINT "gerade"
30 GOTO 10
```

* RND (X)

Nanu, werden Sie jetzt vielleicht sagen, RND (X) gibt es doch auch im normalen Basic. Das ist richtig - aber haben Sie einmal probiert, mit RND Zufallszahlen, sagen wir zwischen 1 und 49 zu erzeugen? Ja, nun dann wissen Sie ja auch, wie kompliziert das ist.

Hier hilft Ihnen EXBASIC LEVEL II. Für Werte X kleiner als 2 funktioniert RND (X) auch in EXBASIC LEVEL II noch ganz normal wie gewohnt. Aber sobald X größer/gleich 2 ist, erzeugt der Befehl RND (X) ganzzahlige (!) Zufallszahlen zwischen 1 und X.

Beispiele:

```
RND (6)  erzeugt Zufallszahlen zwischen 1 und 6
RND (49) erzeugt Zufallszahlen zwischen 1 und 49
RND (100)-1 liefert Zahlen zwischen 0 und 99
RND (21)+9 liefert Zahlen zwischen 10 und 30
```

Beispielprogramm:

```
10 INPUT "höchste Zahl"; Z
20 PRINT RND (Z) : GOTO.
```

Tippen Sie das kurze Programm einmal ein und probieren Sie es aus. Sie sehen dann auch sehr gut, daß RND in EXBASIC LEVEL II nur ganze Zahlen liefert, also Zahlen ohne Dezimalteil (Nachkommateil). Benötigen Sie jedoch Zufallszahlen mit Nachkommastellen, so benutzen Sie einfach folgende Formel:

1+RND(1)*X erzeugt Zahlen zwischen 1 und X mit Dezimalteil

Wir haben auch die Möglichkeit, mit RND Zufallszahlen zwischen einer unteren und einer oberen Grenze zu erzeugen (und nicht nur zwischen 1 und einer oberen Grenze). Die allgemeine Formel für ganze Zahlen dafür lautet:

A-1+RND(B-A+1) erzeugt Zahlen zwischen A und B (ganzz.)

Für Zahlen mit Dezimalteil gilt:

A+RND(1)*(B-A) erzeugt Zahlen zwischen A und B (dezimal)

* HEX$ (X)

HEX$ (X) wandelt die Zahl X in den entsprechenden hexadezimalen String um. Dies ermöglicht Umrechnungen zwischen dem 10er- und dem 16er- Zahlensystem.
Während wir ja im normalen Leben durchweg im 10er System rechnen (schließlich haben wir zehn Finger), ist es für Computeranwendungen eben hin und wieder ganz sinnvoll, im 16er System (Hexadezimalsystem) zu arbeiten. Und zwar deswegen, weil sich 16 als eine Potenz von 2 darstellen läßt (2 hoch 4 = 16), während dies bei 10 nicht gegeben ist.

So arbeitet zum Beispiel der in den Computer eingebaute Terminal Interface Monitor TIM ausschließlich im hexadezimalen Zahlensystem. Hier bietet sich natürlich HEX$ für Umrechnungen an.

Der entstehende String ist entweder 2 Stellen lang (wenn X kleiner als 256 ist) oder 4 Stellen lang (für X größer 255 und kleiner 65536). Bei noch größeren Werten von X wird ein ?ILLEGAL QUANTITY ERROR gemeldet.

* DEC ("string")

DEC ist die Umkehrfunktion zu HEX$. Der gegebene String wird als Hexzahl interpretiert und in die entsprechende Dezimalzahl umgewandelt.

Der String muß exakt zwei oder vier Stellen lang sein, andernfalls erfolgt die Fehlermeldung ?ILLEGAL QUANTITY ERROR.

LEVEL II BASIC Befehle

Sie haben jetzt zwar schon eine ganze Reihe von Möglichkeiten kennengelernt, die Ihnen EXBASIC LEVEL II bietet, aber im Grunde haben wir bisher nur die Hilfsfunktionen und Befehle für spezielle Anwendungen wie Graphik oder Mathematik besprochen.

In diesem Kapitel wollen wir uns nun mit den Befehlen des weltweiten LEVEL II Standards befassen, die das Arbeiten in dieser hohen Programmiersprache zum Vergnügen werden lassen. Sie werden staunen, was Ihnen EXBASIC LEVEL II alles zu bieten hat!

Die Möglichkeiten sind wirklich so vielfältig, daß Sie sie zu Anfang gar nicht alle werden nutzen können. Aber mit der Zeit werden Sie sich immer mehr an EXBASIC LEVEL II und die damit möglichen Programmiertechniken gewöhnen, und feststellen, daß es damit wirklich nicht schwer ist, ein perfekter Programmierer zu werden. Also, los geht's!

* ELSE

Sicher ist Ihnen hinreichend bekannt, wie Abfragen nach bestimmten Bedingungen in Basic programmiert werden können. Nehmen wir einmal an, es soll eine Zahl, die wir eingeben wollen, daraufhin geprüft werden, ob sie größer als 10 ist oder nicht. Der Computer soll eine entsprechende Meldung ausdrucken. Das ist zugegebenermaßen ein recht simples Beispiel, aber es zeigt doch recht deutlich, wie solche Abfragen in Basic programmiert werden.

```
100 INPUT "Geben Sie eine Zahl zwischen 1 und 20 ein!"; Z
110 IF Z > 10 THEN PRINT "Zahl > 10"
120 IF Z <= 10 THEN PRINT "Zahl <= 10"
130 END
```

So sieht das also in herkömmlichem Programmierstil aus. Es sind zwei IF...THEN Abfragen nötig. Selbst ein geschickter Programmierer könnte in normalem Basic höchstens schreiben:

```
100 INPUT "Geben Sie eine Zahl zwischen 1 und 20 ein!"; Z
110 IF Z > 10 THEN PRINT "Zahl > 10" : GOTO 130
120 PRINT "Zahl <= 10"
130 END
```

In EXBASIC LEVEL II sieht dieses Programm nun folgendermaßen aus:

```
100 INPUT "Geben Sie eine Zahl zwischen 1 und 20 ein!"; Z
110 IF Z>10 THEN PRINT "Zahl > 10" ELSE PRINT "Zahl <= 10"
130 END
```

Wie Sie sehen, entfällt Zeile 120 in EXBASIC LEVEL II völlig. Dies wird offensichtlich durch den neuen Befehl ELSE möglich. Wie funktioniert nun ELSE im einzelnen?

ELSE stellt eine Alternativanweisung zu THEN dar. Allgemein können wir schreiben:

```
IF [ausdruck] THEN [anweisung] ELSE [anweisung]
```

Wenn der Ausdruck wahr ist, dann wird die Anweisungsfolge hinter THEN abgearbeitet, wenn er falsch ist, dann diejenige hinter ELSE.

Beispiele:

```
IF A=0 THEN PRINT "Null" ELSE PRINT "ungleich Null"

100 GET X$ : IF X$="" THEN. ELSE PRINT "Taste ";X$

100 GET X$ : IF X$<>"" THEN PRINT "Taste ";X$ ELSE.
```

Das letzte Beispiel zeigt, daß hinter ELSE, genau wie hinter THEN, direkt eine Zeilennummer folgen kann. Bitte erinnern Sie sich daran, daß der Punkt ja nur ein Symbol ist für den Ausdruck "zuletzt abgearbeitete Zeile". Wir könnten das letzte Beispiel also auch folgendermaßen schreiben:

```
100 GET X$ : IF X$<>"" THEN PRINT "Taste ";X$ ELSE 100
```

Selbstverständlich ist es auch in EXBASIC LEVEL II noch möglich, IF...THEN Zeilen zu schreiben, in denen kein ELSE vorkommt. Wir müßten also den allgemeinen Ausdruck eigentlich so formulieren...

```
IF [ausdruck] THEN [anweisung]  ( ELSE [anweisung] )
```

..womit ausgedrückt wird, daß ELSE angegeben werden kann, aber nicht muß. Sie werden aber feststellen, daß ELSE ein so nützlicher Befehl ist, daß Sie bald nicht darauf verzichten werden wollen.

Übrigens können IF..THEN..ELSE Anweisungen auch ineinander geschachtelt werden. Das sieht dann z.B. so aus...

```
IF ... THEN ... IF ... THEN ... ELSE ... ELSE ...
       I               I--------I        I
       I---------------------------------I
```

Selbstverständlich sind auch Schachtelungen mit mehr als zwei IF Anweisungen möglich.
Bei solchen Konfigurationen ist allerdings zu bedenken, daß nur das äußere ELSE entfallen darf (oder alle), weil sich ELSE immer auf das letzte IF bezieht. Dies wiederum ist im Interessere der Eindeutigkeit notwendig. Denn auch wenn eine Programmiersprache einfach sein soll, eindeutig in ihren Formulierungen muß sie dennoch bleiben.

Wenn die Eindeutigkeit gewahrt bleibt, dürfen Sie in EXBASIC LEVEL II übrigens das Wort THEN auch weglassen.

Beispiele:

```
IF A=0 PRINT "A ist Null" ELSE PRINT "A ist ungleich Null"
```

```
100 GET X$ : IF X$<>"" PRINT "Taste ";X$ ELSE.

100 IF A=0 IF B=0 ? "A=B=0" ELSE ? "B<>0" ELSE ? "A<>0"
```

Die Fragezeigen im letzten Beispiel stehen lediglich als Abkürzung für PRINT und haben keinerlei besondere Bedeutung.

THEN muß dann angegeben werden, wenn andernfalls die Eindeutigkeit nicht mehr sichergestellt wäre. So wird z.B. die Zeile...

```
100 IF A=B X=Y
```

..vom Interpreter falsch gelesen, nämlich als...

```
100 IF A=BX= Y
```

..und es entsteht ein ganz anderer Sinn als vorher. Beachten Sie in diesem Zusammenhand, daß ein Leerzeichen (Space) kein gültiges Trennungszeichen ist. Unsere Zeile müßte also richtig lauten...

```
100 IF A=B THEN X=Y     ..oder..     100 IF (A=B) X=Y
```

Achtung: Bedenken Sie bei Wertzuweisungen/Vergleichen in diesem Zusammenhang, daß der Buchstabe E auch Teil einer Zahl sein kann und ggf. vom Interpreter auch so gelesen wird. So ist 'IF A<0 A=0' zwar erlaubt (THEN ist weggelassen), 'IF E<0 E=0' führt jedoch zu der Fehlinterpretation 'IF E<0E =0', wobei '0E' gleich Null angesehen wird!

Während also THEN in den dargelegten Fällen entfallen kann, muß ELSE immer stehen, wenn eine Alternativanweisung zu THEN angegeben werden soll. Ist kein ELSE vorhanden, so fährt das Programm wie üblich bei nicht erfüllter Bedingung zwischen IF und THEN mit der nächsten Programmzeile fort.

Nachfolgend finden Sie einmal ein Programm, das die Eingabe einer dreistelligen Zahl erfordert, aber keinen INPUT-Befehl enthält. Die Ziffer 0 wird nicht angenommen:

```
100 PRINT@ 0, "Geben Sie eine dreistellige Zahl ein: ";
110 GET A$ : IF VAL(A$)=0 THEN. ELSE PRINT A$;
120 GET B$ : IF VAL(B$)=0 THEN. ELSE PRINT B$;
130 GET C$ : IF VAL(B$)=0 THEN. ELSE PRINT C$
140 D$=A$+B$+C$ : PRINT "Sie haben ";D$;" eingegeben!"
```

Und noch ein Beispiel, das die Schachtelung von IF..THEN..ELSE gut zeigt. Das folgende Programm findet immer heraus, welche der beiden Zahlen, die Sie eingeben, die kleinere ist. Wenn möglich, ist THEN nicht ausgeschrieben:

```
10 INPUT "Geben Sie zwei Zahlen ein "; A,B
20 IF A<=B IF A<B PRINTA; ELSE PRINT"keine "; ELSE PRINTB;
30 PRINT "ist kleiner."
```

Ganz schön kurz, nicht wahr?

* RESTORE [zeilennummer]

Mit dem normalen RESTORE-Befehl wird der Zeiger auf die DATA-Liste an den Anfang zurückgesetzt. Es ist nur eine Rücksetzung auf das allererste DATA-Element möglich, was eine starke Einschränkung bei der Programmierung bedeutet.

Diesem Zustand wollten wir, die Entwickler von EXBASIC LEVEL II, Abhilfe schaffen und haben darum den Befehl RESTORE [zeilennummer] kreiert. Damit ist es nämlich möglich, den DATA-Zeiger auf eine bestimmte Zeilennummer zurückzusetzen. Ein Beispiel soll dies verdeutlichen.

```
100 DATA 1,2,3,4,5,6,7,8,9,10
110 DATA 11,12,13,14,15,16
120 RESTORE
130 FOR I= 1 TO 16
140 READ X
150 PRINT X;
160 NEXT I
170 RESTORE 110
180 FOR I= 1 TO 6
190 READ X
200 PRINT X;
210 NEXT I
220 END
```

Dieses Programm führt zu folgendem Ausdruck, wenn Sie es ablaufen lassen:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 11 12 13 14 15 16
I                                    I  I              I
I------------------------------------I  I--------------I
   Schleife von Zeile 130 bis 160       Schleife 180-210
```

Sie sehen, wie der Befehl 'RESTORE 110' in Zeile 110 den Zeiger auf die DATA-Liste wirklich auf Zeile 110 setzt, während der RESTORE-Befehl in Zeile 120 den Zeiger auf den Anfang, also im Beispiel auf Zeile 100 stellt.

Wenn die Zeile, die im Befehl RESTORE [zeilennummer] angegeben ist, gar keine DATA-Zeile ist, so wird der DATA-Zeiger auf die nächstfolgende Zeile gesetzt, die DATA-Elemente enthält.

Beispiel: Wir haben das folgende kurze Programm:

```
10 REM erster Datensatz
20 DATA 1,8,9,45,3,71
30 REM zweiter Datensatz
40 DATA 681,8,2,20,43
50 RESTORE 30
60 READ X : PRINT X
70 END
```

Der Befehle 'RESTORE 30' in Zeile 50 hat dieselbe Wirkung wie sie der Befehl 'RESTORE 40' hätte, da Zeile 30 keine DATA-Zeile ist. Somit liefert 'PRINT X' in Zeile 60 den Wert 681.

Und noch ein Beispiel, wobei diesmal die DATA-Elemente keine Zahlen sind, sondern Strings:

```
10 RESTORE 50
20 READ A$
30 PRINT A$
40 DATA "Zeile 40"
50 DATA "Zeile 50"
```

Das Ergebnis eines Programmlaufs dieses Beispiels ist .... Na, überlegen Sie einmal! Tragen Sie Ihr Ergebnis an der freien Stelle ein. Wenn Sie sich nicht ganz sicher sind, so tippen Sie das kurze Programm ein und probieren Sie es aus.

* ON [numerischer Ausdruck] RESTORE [zeilennummernliste]

Sicher sind Ihnen die Befehle ON...GOTO und ON...GOSUB geläufig. Nun, ON...RESTORE funktioniert ganz ähnlich. Es läßt sich aus einer Liste von Zeilennummern eine berechnen, auf die dann ein RESTORE ausgeführt wird. Die einzelnen Zeilennummern der Zeilennummernliste müssen durch Kommata getrennt werden.

Beispiel:

```
10 ON X RESTORE 50,100,200,300

Für X < 0 erfolgt ?ILLEGAL QUANTITY ERROR
Für X >= 0 und < 1 erfolgt kein RESTORE
Für X = 1 erfolgt ein RESTORE 50
Für X = 2 erfolgt ein RESTORE 100
Für X = 3 erfolgt ein RESTORE 200
Für X = 4 erfolgt ein RESTORE 300
Für X >= 5 erfolgt kein RESTORE
```

Anstelle der Variablen X kann ein beliebiger numerischer Ausdruck stehen. Das Ergebnis des errechneten Ausdrucks wird vom Computer zu einer ganzzahligen Größe umgewandelt, d.h. Nachkommastellen werden ignoriert.

Beispiel:

```
10 ON (A+B) - (C+D) RESTORE 1000,300,150,...
```

Die Anzahl der RESTORE-Zeilennummern ist durch die maximale Zahl von 80 Zeichen pro Zeile begrenzt. Als Alternative bei sehr umfangreichen ON...RESTORE-Befehlen bieten sich daher mehrere ON...RESTORE's an, bei denen die Variable in den weiteren Befehlen dem Programm entsprechend berechnet wird.

Beispiel:

```
5 IF X<0 PRINT "ungültiges X" : STOP
10 IF X<6 ON X RESTORE 100,200,300,400,500 : GOTO 40
20 IF X<11 ON X-5 RESTORE 600,700,800,900,1000 : GOTO 40
30 IF X<16 ON X-10 RESTORE 1100,1200,1300,1400,1500 : GOTO 40
```

* PRINT USING ["format"], [variablenliste]

Durch den EXBASIC LEVEL II Befehl PRINT USING ist es möglich, Zahlen formatiert auf dem Bildschirm oder einem Drucker auszugeben. Damit lassen sich z.B. Zahlen Komma unter Komma drucken, die Anzahl der ausgegebenen Nachkommastellen wird festlegbar und vieles mehr.

Die Formatzeichen sind:

- # Ziffer, Vorkommanullen werden unterdrückt
- * Ziffer, Vorkommanullen werden mit * aufgefüllt
- . Stellung des Dezimalpunktes
- , Stellung des Dezimalkommas
- + das Vorzeichen wird immer ausgegeben
- - nur negatives Vorzeichen wird ausgegeben

Die Vorzeichen-Zeichen '+' und '-' können sowohl vor einer Formatzahl als auch dahinter stehen, dementsprechend wird auch das Vorzeichen gedruckt. Als Dezimalzeichen können Sie wahlweise den Punkt (angelsächsische Schreibweise) oder das Komma (europäisch) verwenden.

Alle Zeichen außer den Formatzeichen im Formatstring werden so ausgedruckt, wie sie dastehen. Nachfolgend finden Sie eine wirklich große Anzahl von Beispielen, weil es so am einfachsten für Sie ist, PRINT USING in all seinen Details kennenzulernen.

Beispiele:

```
PRINT USING "##.##", 12.5            führt zu '12.50'
PRINT USING "###.##", 12.5           führt zu ' 12.50'
PRINT USING "***.**", 12.5           führt zu '*12.50'
PRINT USING "*****.**", 12.5         führt zu '***12.50'
PRINT USING "###.##", -12.5          führt zu ' 12.50'
PRINT USING "+##,##", 12.5           führt zu '+12,50'
PRINT USING "##,##+", 12.5           führt zu '12,50+
PRINT USING "-##,##", 12.5           führt zu ' 12,50'
PRINT USING "##,##-", 12.5           führt zu '12,50 '
PRINT USING "-##,##", -12.5          führt zu '-12,50'
PRINT USING "##.##-", 12.5           führt zu '12.50-'
PRINT USING "###.##-", -.5           führt zu '  0.50-'
PRINT USING "DM ###,##-", 20         führt zu 'DM  20,00 '
PRINT USING "DM ###,##-", -20        führt zu 'DM  20,00-'
PRINT USING "***.** %", 2.89         führt zu '**2.89 %'
PRINT USING "#.#", 3.654             führt zu '3.7'
PRINT USING "#.#+", 3.599            führt zu '3.6+'
PRINT USING "DM**.**+", 2.5          führt zu 'DM*2.50+'
PRINT USING "$ ### ", .5             führt zu '$   0'
PRINT USING "$ ***", .5              führt zu '$ **0'
PRINT USING "$ .##", 2               führt zu '$ .##'
PRINT USING "##.##", 4521.1          führt zu '##.##'
```

Die beiden letzten Beispiele zeigen: ist die angegebene Zahl größer als das spezifizierte Format, so wird anstelle der formatierten Zahl das Format selbst ausgedruckt. Dadurch wird, z.B. bei Listen, wenigstens nicht das gesamte Druckformat gestört.

Wie Sie aus den Beispielen leicht ersehen können, werden Nachkommastellen auf das spezifizierte Format gerundet.

Für formatierte Ausgabe auf einem Drucker muß ein File mit OPEN eröffnet werden. Danach kann mit 'PRINT USING # filenummer' auf den Drucker ausgegeben werden.

Das Format zu PRINT USING kann, wie bisher geschehen, in Anführungszeichen angegeben werden. Wir können aber auch eine Stringvariable benutzen, der wir vorher das entsprechende Format zugewiesen haben.

Beispiel:

```
100 F$= "+###.##"
110 PRINT USING F$, 21.5
```

..führt zu dem Ausdruck...

```
+ 21.50
```

Natürlich können in einem Format auch mehrere Formatzahlen spezifiziert sein.

Beispiel:

```
PRINT USING "##.## plus ##.## =?", 25.3,8.07
```

..führt zu...

```
25.30 plus  8.07 =?
```

Wenn die Anzahl der im Format spezifizierten Formatzahlen (im Beispiel sind es zwei) nicht mit der Anzahl der gegebenen Zahlen (im Beispiel 25.3 und 8.07, also auch zwei) übereinstimmt, so erfolgt die Fehlermeldung ?FORMAT ERROR. Zur gleichen Meldung kommt es bei leerem Formatstring.

Bei einem in sich fehlerhaften Format kommt es i.a. zu einem falschen Ausdruck, nicht jedoch zu einer Fehlermeldung.

Bis jetzt haben wir immer angenommen, daß der Cursor nach dem PRINT USING Ausdruck automatisch eine Zeile tiefer fallen soll. Da das aber gar nicht in allen denkbaren Anwendungsfällen sinnvoll ist, können wir es auch verhindern. Wir fügen dazu am Ende der Variablenliste ein Semikolon an.

Beispiel:

```
PRINT USING "#####.#", 555.77;
PRINT USING "##.## plus ##.## =?", 25.3,8.07;
```

Das Semikolon ganz hinten verhindert ein Fallen des Cursors in die nächste Zeile. Im Grunde genommen wirkt es also genauso wie bei einem normalen PRINT-Befehl. Allerdings darf hier im Unterschied dazu kein Komma stehen (um etwa zur nächsten vortabulierten Position zu gelangen).



REK [numerischer ausdruck] / REK OFF

Sicher ist Ihnen auch schon oft genug unangenehm aufgefallen, daß wir im normalen Basic nur 26 Unterprogrammebenen zur Verfügung haben. Dies bedeutet nicht nur eine Beschränkung in der Benutzung von Unterprogrammtechniken, sondern auch, daß Verfahren wie rekursives Programmieren überhaupt nicht möglich sind. Backtracking o.ä. ist im normalen Basic gar nicht denkbar.

Mit REK werden nun in EXBASIC LEVEL II 64 mal soviel Unterprogrammebenen freigegeben, wie im numerischen Ausdruck spezifiziert sind. Allgemein gilt:

REK X entspricht X*64 Unterprogrammebenen = X*256 Bytes

Beispiele:

REK 4 ergibt 256 Ebenen und belegt 1 kByte
REK 16 ergibt 1024 Ebenen und belegt 4 kByte
REK 64 ergibt 4096 Ebenen und belegt 16 kByte

Sie sehen, daß durch REK dem normalen Basicspeicher Platz weggenommen wird. Wenn Sie also REK 64 eingeben, so haben Sie danach auf einem Computer mit 32 kByte Speicherkapazität nur noch 16 kByte zur Verfügung. Dafür stehen Ihnen allerdings jetzt 4096 Unterprogrammebenen offen.

Beachten Sie, daß durch REK automatisch ein CLR mitausgeführt wird, daß also sämtliche Variablen gelöscht werden.

Der durch REK beanspruchte Speicherplatz wird in der Speicherbelegungsliste bei MEM unter 'REK:' aufgeführt. Um ihn wieder dem Basic zugänglich zu machen, benutzen Sie einfach REK OFF oder REK 0. In beiden Fällen stehen danach wieder nur noch 26 Unterprogrammebenen zur Verfügung und der vorher belegte Speicherplatz ist wieder für Basic frei. Auch REK OFF und REK 0 führen ein CLR mit aus. REK OFF wird auch durch HIMEM und NEW ausgelöst.

Die Wirkung von REK können Sie übrigens mit folgendem kurzen Programm sehr gut sichtbar machen. Lassen Sie es einmal normal laufen und dann einmal, nachdem Sie vorher, sagen wir REK 4 eingegeben haben:

```
10 A= A+1 : PRINT A; : GOSUB 10
```

Nun noch ein Beispielprogramm zur Fakultätsberechnung:

```
10 REK 1 : F=1
20 INPUT "Zahl"; Z
30 GOSUB 50
40 PRINT "Fakultät=";F : END
50 IF Z>1 F=F*Z : Z=Z-1: GOSUB 50
60 RETURN
```

* DISPOSE NEXT (variable) / DISPOSE RETURN / DISPOSE CLR

Um DISPOSE mit all seinen Variationen voll zu verstehen, sollten Sie wissen, daß der Computer intern mit einem sogenannten 'Stack' arbeitet. Wenn wir in Basic eine FOR-NEXT Schleife eröffnen oder ein Unterprogramm mit GOSUB anspringen, so muß sich die Maschine ja irgendwie merken, wo die Schleife anfängt und wie die Schleifenvariable heißt bzw. wo die Rücksprungadresse für das nächste RETURN ist. Zur Abspeicherung all dieser Information dient eben der Stack.

REK wirkt übrigens in der Weise auf den Stack, daß es ihn vergrößert, so daß also mehr Rücksprungadressen für RETURN gespeichert werden können und somit auch mehr Unterprogrammebenen zur Verfügung stehen.

Der Stack arbeitet nun nach dem Prinzip "zuerst herein - zuletzt heraus" (first in - last out). Das bedeutet, die FOR-NEXT-Schleife, die zuletzt eröffnet wurde, muß zuerst wieder geschlossen werden und die Unterprogrammebene, die als letzte angesprungen wurde, muß auch als erste wieder verlassen werden. Wenn Sie das beim Programmieren nicht beachten, tauchen sehr schnell Schwierigkeiten auf.

Beispiele:

```
FOR A= ...     rich-              FOR A=...     falsch
  FOR B=...    tig pro-             FOR B=...   pro-
  NEXT B       gram-              NEXT A        gram-
NEXT A         miert!               NEXT B      miert!

                                r
10 INPUT "Zahl"; Z   f          i   10 INPUT "Zahl"; Z
20 GOSUB 50          a          c   20 GOSUB 50
30 END               l          h   30 IF H THEN 10
50 IF Z=0 THEN 10    s          t   40 END
60 RETURN            c          i   50 H=0 : IF Z=0 THEN H=1
                     h          g   60 RETURN
```

Im ersten Beispiel muß die innerste, also zuletzt eröffnete Schleife als erste geschlossen werden (so wie es links dargestellt ist), im anderen Fall (rechts) kommt es zur Fehlermeldung ?NEXT WITHOUT FOR ERROR, sobald der Computer an das NEXT B stößt. Im zweiten Beispiel ist der Sprung in Zeile 50 (links) nicht erlaubt (THEN 10), da eine Unterprogrammebene nur mittels RETURN verlassen werden darf. Sie sehen, wie im zweiten Beispiel die Einführung einer Hilfsvariablen H notwendig ist, um den gewünschten Effekt zu erzielen (rechts). Wenn Sie jetzt gar mit mehreren Unterprogrammebenen arbeiten, so führt das zu einem beträchtlichen Aufwand ohne jegliche Übersichtlichkeit.

Und genau hier hilft DISPOSE. Es erlaubt nämlich den Stack um jeweils eine offene Schleife (DISPOSE NEXT) oder eine offene Unterprogrammebene (DISPOSE RETURN) zu entleeren. DISPOSE CLR leert den ganzen Stack. Damit ist es z.B. möglich, von einer Unterprogrammebene beliebiger Tiefe direkt in das Hauptprogramm zurückzuspringen.

Beschäftigen wir uns zunächst mit DISPOSE NEXT.
DISPOSE NEXT nimmt die zuletzt geöffnete FOR-NEXT-Schleife vom Stack. Dies erlaubt einen Aussprung aus einer Schleife ohne NEXT.

Beispiel:

```
10 FOR X= 1 TO 55
20 PRINT X
30 IF X=20 DISPOSE NEXT
40 NEXT X
```

Wenn Sie dieses Beispiel starten, dann läuft es bis zu X=20. Sobald aber X den Wert 20 erreicht, so wird DISPOSE NEXT in Zeile 30 ausgeführt. Die Schleife mit der Schleifenvariablen X wird folglich geschlossen und das darauffolgende NEXT X führt zwangsläufig zu einem ?NEXT WITHOUT FOR ERROR. Dieses Beispiel ist also sicher nicht sehr sinnvoll, es zeigt aber sehr gut, wie DISPOSE NEXT wirkt.

Jetzt noch ein sinnvolleres Beispiel:

```
10 INPUT "Anzahl der Eingaben"; N
20 DIM F(N)
30 FOR I= 1 TO N
40 PRINT "Eingabe Nr.";I;":";
50 INPUT F(N)
60 IF F(N)=0 DISPOSE NEXT : GOTO 90
70 NEXT I
80 PRINT "Ende der Eingabe" : END
90 PRINT "Abbruch der Eingabe" : END
```

Es geht darum, eine Reihe von Zahlen in ein Feld (F) einzugeben. Dabei können Sie vorher wählen, wieviel Eingaben zu tätigen sind. Sie können aber auch zwischendurch aus der Eingaberoutine jederzeit aussteigen, indem Sie eine 0 eintippen (Abfrage in Zeile 60). In diesem Fall erscheint die Meldung "Abbruch der Eingabe", ansonsten "Ende der Eingabe". Und für dieses Beispiel ist eben DISPOSE NEXT gut geeignet.

Hinter DISPOSE NEXT darf eine Variable folgen, also z.B. DISPOSE NEXT A. Dies schließt die Schleife mit der Variablen A (und nicht unbedingt nur die zuletzt geöffnete). Genau wie bei NEXT A werden aber auch alle Schleifen innerhalb von A geschlossen. Mehrere Variablen hinter DISPOSE NEXT führen zu einem ?SYNTAX ERROR.

Beispiel:

```
10 FOR A= 1 TO 100
20  FOR B= 1 TO 10
30  IF A=10 AND B=5 DISPOSE NEXT A : GOTO 60
40  NEXT B
50 NEXT A
60 END
```

DISPOSE NEXT A schließt sowohl Schleife A, als auch die innere Schleife B.

DISPOSE RETURN holt eine Unterprogrammebene vom Stack, d.h. es vollführt einen Sprung von einer Ebene zur nächsthöheren, ohne dabei RETURN auszuführen. Damit ist es z.B. möglich, aus einem Unterprogramm auch mittels GOTO in das Hauptprogramm zurückzuspringen.

Beispiel:

```
10 REM Hauptprogramm
20 GOSUB 50
30 PRINT "Hier ist das Hauptprogramm"
40 END
50 REM Unterprogramm
60 DISPOSE RETURN
70 GOTO 30
```

In Zeile 20 erfolgt der Einsprung in das Unterprogramm. Durch DISPOSE RETURN in Zeile 60 erfolgt die Rückkehr auf die Ebene des Hauptprogramms und jetzt ist auch der Sprung in Zeile 70 erlaubt.

Ein ausführlicheres Beispiel:

```
100 REM Menüprogramm
110 DIM A(100)
120 PRINT "Hier ist das Menüprogramm"
130 PRINT "1. Programmstart"
140 PRINT "2. Menü"
150 GET X$: IF X$="1" RUN
160 IF X$="2" GOSUB 900: GOTO 120
170 GOTO 150
900 REM Auswahlroutine
910 PRINT "Bitte wählen Sie:"
920 PRINT "1. Kundenerfassung"
930 PRINT "2. Statistische Auswertung"
940 PRINT "*. zurück ins Menüprogramm
950 GET X$: IF X="*" DISPOSE RETURN: GOTO 120
960 ON VAL(X$) GOSUB 1000,2000
970 GOTO 950
```

Dieses Beispiel zeigt ein Menüprogramm mit mehreren (im Beispiel zwei) Auswahlebenen. Von einer Ebene auf die nächsttiefere wird normal mittels GOSUB gesprungen, von einer Ebene auf die nächsthöhere dagegen mit DISPOSE RETURN und anschließendem GOTO.

Bedenken Sie bitte bei all diesen Beispielprogrammen, daß sie lediglich die Eigenheiten der gerade besprochenen neuen EXBASIC LEVEL II Befehle zeigen sollen und nicht mehr. Wir sind uns selbstverständlich voll darüber im klaren, daß diese Beispiele keinesfalls ausgereift sind. Lassen Sie sich also bitte nicht zu dem Schluß hinreißen, daß wir nicht besser programmieren könnten, wir wollen die kleinen Beispiele lediglich so einfach wie möglich halten. Auch wenn das vielleicht hin und wieder dem Feingefühl des einen oder anderen unter den verehrten Lesern entgegensteht. Fertige Beispielprogramme finden Sie zu Genüge am Ende dieses Anleitungsbuches.

DISPOSE CLR schließlich leert den gesamten Stack auf einmal. Es werden also sämtliche Schleifen geschlossen und alle Unterprogrammebenen verlassen. Variablen werden durch DISPOSE CLR nicht gelöscht.

Beispiel:

```
10 TI$= "000000"
20 GOSUB 100
30 PRINT "Zeit="; TI$
40 END
100 FOR A= 1 TO 10
110  FOR B= 1 TO 50
120  GET X$ : IF X$="*" DISPOSE CLR : GOTO 20
130  NEXT B
140 NEXT A
150 RETURN
```

Das Programm mißt die Zeit, die für das Durchlaufen der zwei Schleifen im Unterprogramm notwendig ist. Mit der Taste '*' wird das Programm jedesmal neu gestartet, ohne daß auch die Zeit genullt wird. Ohne DISPOSE CLR müßte Zeile 120 wie folgt aussehen und eine weitere Zeile hinzukommen...

```
120 GETX$:IFX$="*"THENB=50:A=10:NEXTB,A:H=1:RETURN
25 IFH=1THENH=0:GOTO20
```

Da erspart DISPOSE CLR doch einiges, nicht wahr?

* INPUTLINE ("text";) [stringvariable]

INPUTLINE ist eine Variation des normalen INPUT-Befehls. Wichtigster Unterschied ist, daß bei INPUTLINE wirklich alles eingegeben werden darf, also auch Anführungszeichen, Kommata, Doppelpunkte, Steuerzeichen wie z.B. Cursorsteuerungszeichen u.ä., eben alles ohne Ausnahme.

Daraus ergibt sich konsequenterweise, daß zu INPUTLINE im Gegensatz zu INPUT nur eine einzige Variable mitgegeben werden darf, in die dann alles eingelesen wird. Die Variable kann also auch mehrere Anführungszeichen u.a. nach INPUTLINE enthalten. Es muß sich um eine Stringvariable handeln, sonst wird ein ?TYPE MISMATCH ERROR gemeldet.

Beispiele für richtige INPUTLINE-Anweisungen:

```
INPUTLINE "Name"; N$
INPUTLINE X$
```

Beispiele für falsche INPUTLINE-Anweisungen:

```
INPUTLINE "Zahl"; Z                    keine Stringvariable
INPUTLINE "Name, Vorname"; N$,V$       mehr als eine Variable
INPUTLINE; X$                          Semikolon ohne Text
```

Genau wie bei INPUT ist auch bei INPUTLINE kein Break durch Drücken der STOP-Taste möglich. Eine leere Eingabe bricht das Programm nicht ab. Mit INPUTLINE können bis zu 79 Zeichen eingelesen werden.

Beispiel:

```
10 INPUTLINE "Text"; T$
20 PRINT T$
```

Nun kann in Zeile 10 ein beliebiger Text in Anführungszeichen eingegeben werden, und zwar inklusive Cursorsteuerungs-zeichen, Bildschirmlöschen, reversen Zeichen etc., der dann in Zeile 20 ausgedruckt wird.

* INPUTFORM ("text";) [stringvariable] (,länge)

INPUTFORM können wir zunächst einmal ähnlich wie INPUTLINE benutzen, indem wir keine Länge spezifizieren. Im Gegensatz zu INPUTLINE erlaubt INPUTFORM aber keine Eingabe von Cursorsteuerungszeichen, reversen Zeichen etc., wohl aber von Anführungszeichen, Kommata, Doppelpunkten und geshifteten Zeichen. Zu INPUTFORM darf deshalb logischerweise nur eine Stringvariable mitgegeben werden.

Beispiele:

```
INPUTFORM "Bitte geben Sie hier Ihren Namen ein"; N$
INPUTFORM X$
```

Weitere Unterschiede zu INPUTLINE sind: Bei INPUTFORM erscheint anstelle des blinkenden Cursors ein feststehendes Unterstreichungszeichen. Die DEL-Taste wirkt wie üblich, INST und Cursorsteuerung funktionieren nicht. Dafür kann durch Drücken der Taste 'Pfeil nach links' die gesamte bisherige Eingabe gelöscht werden (DEL erlaubt ja nur Einzelzeichen-löschung).

Eine INPUTFORM-Eingabe kann, genau wie INPUTLINE, nicht durch Drücken der Taste STOP abgebrochen werden.

Wenn Sie in einem Programm grundsätzlich verhindern wollen, daß es durch die STOP-Taste abgebrochen werden kann, so geben Sie dazu...

```
DOKE 144,58929  (Serien 2001/3001)
DOKE 144,58456  (Serien 4001/8001)
```

..ein. Durch diese Befehle wird die STOP-Taste inaktiviert. Gehen Sie damit vorsichtig um, denn in diesem Modus können Sie den Programmlauf oder ein Listing nicht unterbrechen! Um die STOP-Taste wieder zu aktivieren, geben Sie ON ein (ON ist auch programmierbar). ON schaltet gleichzeitig auch Repeat-funktion (Serien 2001/3001/4001) und den Fehlermodus ein.

Wenn Sie die Funktion der STOP-Taste wieder einschalten wollen, ohne gleichzeitig Repeat und Fehlermodus mit anzusprechen, so benutzen Sie statt ON einfach OFF. Auch nach OFF funktioniert die STOP-Taste wieder normal. Prinzipiell wäre übrigens auch BASIC dazu geeignet. Allerdings steht dann EXBASIC LEVEL II nicht länger zur Verfügung.

Wenn Sie bei INPUTFORM eine leere Eingabe vornehmen, d.h. wenn Sie die RETURN-Taste drücken, ohne vorher etwas eingetippt zu haben, so behält die Eingabevariable (im Beispiel N$ bzw. X$) ihren alten Inhalt. Zu einem Programmabbruch kommt es in diesem Fall, im Gegensatz zum normalen INPUT, nicht. Dadurch ist ein nicht gewolltes Verlassen eines Programms durch versehentliches Drücken der RETURN-Taste ohne eine Eingabe ausgeschlossen.

Zu INPUTFORM kann außer all den bisher genannten Eigenschaften eine Eingabelänge mit angegeben werden. INPUTFORM nimmt dann höchstens so viele Zeichen an, wie in der Länge spezifiziert werden. Die Eingabelänge wird durch ein Komma von der Variablen getrennt und darf Werte zwischen 1 und 79 haben. INPUTFORM löscht genau so viele Zeichen, wie eingegeben werden können.

Beispiele:

```
INPUTFORM "Name"; N$ ,12   Eingabe mit bis zu 12 Stellen
INPUTFORM "J/N"; X$ ,1     Ja/Nein Entscheidung (1 Stelle)
INPUTFORM "Text"; E$ ,79   maximale Eingabelänge ist 79
INPUTFORM "Text"; T$       keine Eingabelänge spezifiziert
```

Noch ein Hinweis zu einem INPUTFORM-Befehl in die letzte Bildschirmzeile:

INPUTFORM weiß in Abhängigkeit von der Eingabelänge, die Sie angegeben haben, und von der Länge des mitgegebenen Textes, ob ein 'scrolling' (Hochschieben des Bildschirms) notwendig ist oder nicht. Entsprechend wird es gleich zu Beginn des INPUTFORM-Befehls durchgeführt oder gar nicht.

Beispiel:

```
INPUTFORM "Eingabe"; E$ ,35
```

..erfordert 'scrolling', da 7 ("Eingabe") plus 35 größer als 40 ist und somit die gesamte Eingabe nicht mehr in eine Zeile paßt. Dies gilt im Beispiel nicht für die Serie 8001, deren Bildschirmbreite ja 80 statt 40 Zeichen beträgt.
Diese Überlegungen gelten durchweg selbstverständlich nur, wenn der INPUTFORM-Befehl in die letzte Bildschirmzeile gelegt wird. Andernfalls kommt es nie zu einem 'scrolling'.

INPUTLINE und INPUTFORM können nicht im Direktmodus benutzt werden, andernfalls kommt es zur Fehlermeldung ?ILLEGAL DIRECT ERROR, genau wie bei INPUT. Schließlich ist ein Eingabebefehl nur innerhalb eines Programmes sinnvoll.

* DEF USR = [adresse]

Um ein im Speicher stehendes Maschinenprogramm aufzurufen, haben wir mehrere Möglichkeiten zur Verfügung.
Einmal ist der Einsprung mit SYS [adresse] möglich. Das hat aber den Nachteil, daß zu SYS kein Parameter mitgegeben werden kann.

Zum anderen kann ein Maschinenprogramm mit USR (X) aufgerufen werden. Dabei darf wenigstens ein Argument X in Klammern mitgegeben werden. Das Argument wird im sog. 'floating point accumulator' abgelegt und kann dort vom Maschinenprogramm aus benutzt werden. Die Einsprungadresse für USR müssen wir vor der erstmaligen Benutzung im sog. USR-Vektor festlegen, im Gegensatz zu SYS wird sie ja dem Befehl nicht mitgegeben. Wird keine USR-Adresse explizit definiert, so ist der USR-Vektor auf ?ILLEGAL QUANTITY ERROR gerichtet.
Zur Neufestlegung dient nun DEF USR.

Beispiele:

```
DEF USR = 826          definiert den zweiten Kassettenpuffer
DEF USR = 37100        definiert EXBASIC LEVEL II Einsprung
```

Die durch DEF USR festgelegte Adresse kann jederzeit mit...

```
PRINT DEEK (1)
```

..abgefragt werden. Die Bedeutung von DEEK werden wir etwas weiter hinten besprechen.

* DEF CALL = [adresse]

In EXBASIC LEVEL II gibt es neben den schon erwähnten Einsprungmöglichkeiten in Maschinenprogramme, SYS und USR, zusätzlich den Befehl CALL. Mit CALL ist es möglich, nicht nur einen, wie bei USR, sondern eine ganze Reihe von Parametern mitzugeben. Wir werden noch im einzelnen darauf zu sprechen kommen.

Genau wie der USR-, so muß auch der CALL-Vektor vor der Benutzung von CALL mit DEF CALL festgelegt werden. Nach dem Initialisieren von EXBASIC LEVEL II ist CALL auf EXBASIC LEVEL II selbst gerichtet. Um das zu ändern, geben Sie einfach DEF CALL ein.

Beispiele:

```
DEF CALL = 826         definiert den zweiten Kassettenpuffer
DEF CALL = 37100       definiert EXBASIC LEVEL II Einsprung
```

Auch die durch DEF CALL festgelegte Adresse ist abfragbar. Geben Sie dazu...

```
PRINT DEEK (1021)
```

..ein.

* CALL (parameterliste)

Mit CALL können Sie eigene Befehle in EXBASIC LEVEL II implementieren. Genau wie die EXBASIC LEVEL II Befehle DEF USR und DEF CALL ist CALL praktisch für Sie als Programmierer nur interessant, wenn Sie auch in Assembler tätig sind. Falls dem nicht so sein sollte, so empfehlen wir Ihnen, den folgenden Abschnitt trotzdem einmal durchzulesen, aber seien Sie nicht enttäuscht, wenn Sie nicht alles verstehen. Schließlich ist Assembler eine eigene Programmiersprache, die zu erlernen ungefähr ebensoviel Zeit erfordert, wie Basic zu lernen.

CALL ist ein EXBASIC LEVEL II Befehl zum Aufruf eines Maschinenprogrammes, bei dem beliebig viele Parameter mitgegeben werden können. Die Parameter müssen von Ihrem eigenen Assemblerprogramm selbst gelesen und verarbeitet werden. Dadurch sind Sie in der Anwendung von CALL völlig frei.

Genau wie für USR, so existiert auch für CALL ein Vektor, der bestimmt, ab welcher Adresse Ihr Maschinenprogramm mit CALL aufgerufen wird. Der CALL-Vektor liegt in den Speicherzellen 1021/1022. Definiert wird er am einfachsten mit DEF CALL.

Beispiel:

```
Serien 2001/3001:  DEF CALL = 55047
Serien 4001/8001:  DEF CALL = 51546
```

Nach dieser Festlegung des CALL-Vektors entspricht CALL dem Befehl POKE des normalen Basics. Allerdings müssen die Parameter zu CALL immer in Klammern angegeben werden (anders als beim normalen POKE).

Beispiel:

```
CALL (2000,100) entspricht nun dem Befehl POKE 2000,100
```

Damit Ihnen der Anschluß Ihres Maschinenprogramms an CALL nicht so schwer fällt, haben wir Ihnen anschließend die Adressen der wichtigsten Routinen zum Lesen der Parameter aufgelistet:

Bedeutung:	2001/3001	4001/8001
lese Ausdruck	52383	48536
lese Ausdruck und prüfe auf numerisch	52363	48516
prüfe String	52368	48521
lese Komma	52728	48885
konvertiere float. point in Integer	53402	49898
Integer in float. point	53869	50364
lese 0<= Ausdruck <=255 (X-Register)	54904	51412

Damit Sie einmal sehen können, wie CALL funktioniert, anbei ein etwas ausführlicheres Beispiel. Das Assemblerlisting ist speziell für die Serie 2001/3001, die für die Serien 4001 und 8001 notwendigen kleinen Änderungen sind aber direkt darunter angegeben.

```
Adr Assem.befehl     Hex-Code        Kommentierung

826 JSR  54904       20 78 D6        ;Schrittweite einlesen
829 TXA              8A              ;Statusregister setzen
830 BEQ  868         F0 24           ;Rücksprung bei Weite 0
832 STA  70          85 46
834 JSR  52728       20 F8 CD        ;Kommatest
837 JSR  54904       20 78 D6        ;Zeichen lesen
840 LDY #0           A0 00           ;Start
842 LDA #128         A9 80
844 STY  64          84 40
846 STA  65          85 41
848 TXA              8A              ;Zeichen im Akku..
849 STA (64),Y       91 40           ;..ausdrucken
851 LDA  64          A5 40
853 CLC              18
854 ADC  70          65 46           ;Weite addieren
856 STA  64          85 40
858 BCC  848         90 F4
860 INC  65          E6 41
862 LDA  65          A5 41
864 CMP #132         C9 84           ;Ende erreicht?
866 BCC  848         90 EC
868 RTS              60              ;Rücksprung zu EXBASIC
```

```
Änderungen Serie 8001:              Änderungen Serie 4001:

826 JSR  51412    20 D4 C8          826 JSR  51412    20 D4 C8
834 JSR  48885    20 F5 BE          834 JSR  48885    20 F5 BE
837 JSR  51412    20 D4 C8          837 JSR  51412    20 D4 C8
864 CMP #136      C9 88
```

Das ist das Assemblerlisting. Was das kurze Programm tut? Nun, es bedruckt den Bildschirm mit einem von Ihnen bestimmten Zeichen in einer von Ihnen festgelegten Schrittweite. Und selbstverständlich wird es über CALL aufgerufen. Das Format für den neuen Befehl lautet...

```
CALL (schrittweite,zeichen)
```

Doch bevor (!) wir das Proramm zum ersten Mal aufrufen, müssen wir es natürlich erst einmal eintippen. Sie können sich dazu des Terminal Interface Monitors TIM bedienen (Einsprung mit GO). Der für TIM benötigte hexadezimale Code ist in der Spalte 'Hex-Code' gegeben. Wenn Sie EXBASIC LEVEL II für die Serie 8001 besitzen, so können Sie auch EXMON zur Eingabe benutzen (Einsprung ebenfalls mit GO).

Wir wollen aber im folgenden noch eine andere Art besprechen, ein in Hex-Code vorliegendes Maschinenprogramm mit EXBASIC LEVEL II zu verbinden.

Wir schreiben den Maschinencode hexadezimal in die DATA-Zeilen eines EXBASIC LEVEL II Programmes und bringen ihn beim Programmlauf in die entsprechenden Speicherzellen mittels POKE hinein:

```
10 REM Beispielprogramm für CALL / DEF CALL
100 REM Serie eingeben
110 INPUTFORM "Serie 2001/3001/4001/8001";S$,4
120 S=VAL(LEFT$(S$,1))
130 IF S<>2 AND S<>3 AND S<>4 AND S<>8 THEN 100
200 REM Hex-Code einlesen und POKEn
210 READ X$ : IF X$="*" THEN 300
220 POKE 826+I, DEC(X$)
230 I= I+1
240 GOTO 200
250 REM Hex-Code (Abschluß mit *)
260 DATA 20,78,D6,8A,F0,24,85,46,20,F8,CD,20
270 DATA 78,D6,A0,00,A9,80,84,40,85,41,8A,91
280 DATA 40,A5,40,18,65,46,85,40,90,F4,E6,41
290 DATA A5,41,C9,84,90,EC,60,*
300 REM Serienunterscheidung
310 IF S=2 OR S=3 THEN 400
320 DOKE 827, 51412: DOKE 835, 48885
330 DOKE 838, 51412
340 IF S=8 POKE 865,136
400 REM CALL-Vektor definieren
410 DEF CALL= 826
500 REM Aufrufformat drucken
510 PRINT "Aufruf mit CALL (weite,zeichen)"
600 END
```

Wenn Sie dieses Programm eingetippt haben und einmal laufen lassen, so transportiert es den HEX-Code in den zweiten Kassettenpuffer (dez. 826 bis 1017) und setzt den CALL-Vektor auf die Anfangsadresse 826.

Sollten Sie einmal vor dem ersten Aufruf eines Programms mit CALL vergessen, den CALL-Vektor einzustellen, so zeigt dieser automatisch auf EXBASIC LEVEL II. Das macht aber nichts, da durch einen solchen Einsprung in EXBASIC LEVEL II der Basicspeicher nicht gelöscht wird. Es gehen Ihnen also weder Programme noch Daten verloren.

Um das Beispielprogramm auszuprobieren, geben Sie nun - nachdem Sie es einmal mit RUN gestartet haben - am besten einmal...

```
CALL (10,42)
```

..ein. Probieren Sie auch andere Zahlenkombinationen aus. Bedenken Sie, der erste Parameter steht für die Schrittweite, der zweite für das Zeichen selbst. Beide können Werte zwischen 0 und 255 annehmen, allerdings führt die Schrittweite 0 zu keinem Ausdruck.

Die Anwendungsmöglichkeiten für CALL sind äußerst vielfältig. So dient CALL auch zum Verbinden der SOFTMODULE mit EXBASIC LEVEL II.

* DOKE [adresse], [numerischer ausdruck]

Das EXBASIC LEVEL II Wort DOKE steht für "Doppelbyte-POKE". Mit POKE können wir den Inhalt einer Speicherzelle gezielt verändern. POKE bezieht sich immer nur auf eine einzige Adresse, es sind somit nur Inhalte zwischen 0 und 255 erlaubt (2 hoch 8). Die CPU 6502 ist nun einmal ein 8 Bit Mikroprozessor.

Beispiele:

```
POKE 826, 96   "POKEt" den Wert 96 in Adresse 826
POKE 1, 255    höchster bei POKE zugelassener Wert (255)
```

Während POKE nur eine Speicherzelle beeinflußt, nämlich diejenige, deren Adresse angegeben ist, spricht DOKE die spezifizierte und die darauffolgende gleichzeitig an. Die mit DOKE angegebene Zahl kann zwischen 0 und 65535 liegen (2 hoch 16 ist 65536) . Sie wird gemäß allgemeiner Normung in zwei Bytes zerlegt (niederwertiges (LSB) und höherwertiges (MSB) Byte). Wenn der Wert kleiner als 256 ist, so wird das höherwertige Byte als 0 betrachtet.

Beispiel:

```
DOKE 1, 826 entspricht den Befehlen...

            POKE 1, 58    (less significant byte LSB)
            POKE 2, 3     (most significant byte MSB)
```

..,weil 58 + 3*256 gleich 826 (Anfang des zweiten Kassettenpuffers) ist.

Allgemein gilt für DOKE:

```
DOKE adresse,wert
```

..enspricht den Befehlen...

```
POKE adresse, wert-INT(wert/256)*256
POKE adresse+1, wert/256
```

Beispiele:

```
DOKE 1, 826          legt USR-Vektor auf Adresse 826
DOKE 1021, 826       legt CALL-Vektor auf 826 fest
DOKE 144, 58929      inaktiviert STOP (2001/3001)
DOKE 144, 58456      inaktiviert STOP (4001/8001)
```

Anstelle der ersten beiden DOKE-Befehle können Sie in EXBASIC LEVEL II natürlich besser 'DEF USR = 826' und 'DEF CALL = 826' verwenden.

Ebenso wie POKE ist auch DOKE nur sinnvoll, wenn es auf eine Speicherzelle angewandt wird, die im RAM-Bereich liegt. Bei POKE oder DOKE in ROM oder EPROM erfolgt zwar keine Fehlermeldung, der Inhalt der angegebenen Adresse wird aber selbstverständlich nicht geändert.

* DEEK (adresse)

DEEK ist der zu DOKE zugehörige Befehl. DEEK steht für "Doppelbyte-PEEK". Während DOKE es Ihnen erlaubt, den Inhalt zweier Speicherstellen mit einem Befehl zu ändern, ermöglicht DEEK den Inhalt zweier Speicherstellen auf einmal zu lesen. Entsprechend üblicher Konvention werden niederwertiges (LSB) und höherwertiges (MSB) Byte zu einem dezimalen Wert zusammengefaßt.

Beispiele:

```
PRINT DEEK (1)          druckt den USR-Vektor aus
PRINT DEEK (1021)       druckt den CALL-Vektor aus
DOKE 900, DEEK(900)+2 erhöht Adresse in 900/901 um 2
PRINT HEX$(DEEK(900)) liefert Adresse in 900/901 in hex
```

Allgemein ist..

```
PRINT DEEK (adresse)
```

..dasselbe wie...

```
PRINT PEEK(adresse) + PEEK(adresse+1) * 256
```

Sowohl DOKE als auch DEEK arbeiten mit dezimalen Parametern, zur Umrechnung stehen Ihnen die Funktionen DEC (hexadez. in dez.) und HEX$ (dez. in hexadez.) zur Verfügung.

* VARPTR (variablenname)

Diese Funktion gibt die Adresse an, wo eine bestimmte Variable im Arbeitspeicher vom Betriebssystem abgelegt worden ist. Falls die betreffende Variable noch gar nicht benutzt wurde, wird sie durch VARPTR angelegt.

Beispiele:

```
A=5:? VARPTR(A)     zeigt, wo Variable A abgespeichert ist
X$="XY":?VARPTR(X$) druckt aus, wo X$ gespeichert ist
```

VARPTR liefert die Adresse des ersten Bytes, ab welchem der zur Variablen gehörende Wert gespeichert ist (Byte 3 in unterer Tabelle). Variablen werden wie folgt abgelegt:

Byte	1	2	3	4	5	6	7
real	n.g.	n.g.	in Byte 3 bis 7 steht der Variab.wert				
%	ges.	ges.	MSB	LSB	0	0	0
$	n.g.	ges.	länge	zeiger auf text		0	0
FN	ges.	n.g.	prgzeiger anf.		variabl.-zeiger		-

Der Variablenname steht in Byte 1 und 2. Die Eintragungen 'nicht gesetzt' (n.g.) und 'gesetzt' (ges.) beziehen sich dabei auf Bit 7 (gezählt 0...7). Bei Stringvariablen verweist der Zeiger in Byte 4 und 5 auf die Adresse, wo der Text steht.

* SPACE [spalte lo,zeile lo,spalte ru, zeile ru] <,wert>

Zu den Abkürzungen: 'lo' steht für "links oben", 'ru' für "rechts unten". Anstelle des Wertes in spitzen Klammern kann jeder beliebige numerische Ausdruck stehen.

SPACE mit Angabe zweier Punkte füllt das durch diese gegebene Rechteck auf dem Bildschirm mit dem Zeichen, dessen Bildschirmcode durch den numerischen Ausdruck gegeben ist. Wenn Sie keine Codeangabe machen, so wird das spezifizierte Feld gelöscht (ASCII 32).

Welche beiden Punkte Sie festlegen müssen, finden Sie noch einmal auf folgender Skizze verdeutlicht. A ist der Punkt links oben, B derjenige rechts unten. Beachten Sie, daß die Zeilennumerierung von 0 bis 24, die Spaltennumerierung von 0 bis 39 (40 Zeichen Bildschirmbreite) bzw. bis 79 (80 Zeichen Bildschirmbreite) geht.

```
A---------------------
I  durch SPACE de-   I
I  finierter Bild-   I
I  schirmausschnitt  I
---------------------B
```

Der Code zu SPACE ist nicht der normale 7-bit ASCII-Code, der über ASC abfragbar ist. Vielmehr handelt es sich um den Bildschirmcode, der einer 6-bit Codierung unterliegt.

Um nun den Code zu einem bestimmten Zeichen festzustellen, gehen Sie am besten wie folgt vor (Sie können natürlich auch mit SPACE probieren):
Löschen Sie den Bildschirm und bringen Sie das Zeichen, dessen Code Sie wissen wollen, in die linke obere Ecke des Bildschirms (HOME Position). Wenn Sie jetzt ein paar Zeilen tiefer...

```
PRINT PEEK (32768)
```

..eingeben und die RETURN-Taste drücken, so erhalten Sie den gesuchten Code. Er kann zwischen 0 und 255 liegen.

Um den für SPACE erforderlichen 6-bit ASCII-Code näher kennenzulernen, können Sie sich auch des folgenden kurzen Programms bedienen:

```
100 PRINT "Bildschirm löschen"
110 PRINT "HQQQQ" :' H=HOME, Q=Cursor n. unten
120 INPUT "Code=";X
130 IF X<0 OR X>255 THEN 100
140 POKE 32768,X
150 GOTO 110
```

Geben Sie nach Programmstart einen beliebigen Code zwischen 0 und 255 ein, drücken Sie RETURN und schon erscheint das dazugehörige Zeichen in der linken oberen Bildschirmecke. Bitte geben Sie in den Zeilen 100 und 110 die entsprechenden Steuerzeichen ein und nicht etwa den Text.

Beispiele zu SPACE:

```
SPACE 0,0,39,24  löscht den ganzen Schirm (40 Zeichen)
SPACE 0,0,79,24  löscht den Bildschirm (80 Zeichen/Zeile)
SPACE 0,10,39,10 löscht Zeile 10 (40 Zeichen Breite)
SPACE 0,10,79,10 löscht Zeile 10 (80 Zeichen Breite)
SPACE 0,10,39,12 löscht von Zeile 10 bis 12 (40 Zeichen)
SPACE 0,10,79,12 löscht von Zeile 10 bis 12 (80 Zeichen)
SPACE 0,0,19,24  löscht linken Bildschirmteil
SPACE 20,0,20,24 löscht Spalte 20
SPACE 12,0,18,24 löscht von Spalte 12 bis Spalte 24

SPACE 2,5,10,12,42  beschreibt ein Feld mit '*'
SPACE 2,5,10,12,32  löscht dasselbe Feld wieder
SPACE 2,5,10,12     löscht ebenfalls (32 ist Ersatzwert)
SPACE 2,5,10,12,160 füllt das Feld voll aus (RVS Space)
```

Wie Sie sehen, ist SPACE wirklich äußerst vielfältig anwendbar; praktisch überall dort, wo es auf eine saubere Gestaltung des Bildschirms ankommt.

Schauen Sie sich noch folgendes kurzes Programm an. Es entwirft eine Umrahmung des Bildschirms nach Ihrem Muster:

```
100 BS= 39 :REM 40 Zeichen (2001/3001/4001)
100 BS= 79 :REM 80 Zeichen (Serie 8001)

110 INPUT "Code für Umrahmung";X
120 IF X<0 OR X>255 THEN 110
130 SPACE 0,0,BS,0,X
140 SPACE 0,24,BS,24,X
150 SPACE 0,0,0,24,X
160 SPACE BS,0,BS,24,X
```

Damit Ihnen die Benutzung von SPACE leichter fällt, haben wir Ihnen einmal die wichtigsten Zeichen im 6-bit ASCII Code aufgelistet. Beachten Sie aber, daß auch der Bildschirmmodus (Groß/Klein oder Graphik) eine entscheidende Rolle spielt.

```
0          = '@'
1...26     = 'a' bis 'z'
32         = Leerzeichen
33         = '!'
42         = Asterisk '*'
48...57    = '0' bis '9'
65...90    = 'A' bis 'Z'
129...154  = RVS 'a' bis 'z'
160        = voller Cursor (RVS Leerzeichen)
176...185  = RVS '0' bis '9'
193...218  = RVS 'A' bis 'Z'
```

Ein Hinweis zur Systematik: Jedes reverse Zeichen hat im 6-bit ASCII Code einen um genau 128 höheren Wert als das dazugehörige normale Zeichen.

Beispiel:

42 ist der Code für '*', 42+128= 170 der für RVS '*'

* STRING$ (anzahl,string oder numerischer ausdruck)

STRING$ erzeugt einen String mit der Länge 'anzahl' (zwischen 0 und 255), der aus Zeichen besteht, die durch 'string' oder 'numerischer ausdruck' vorgegeben sind. Wenn ein String angegeben ist, so wird das erste in ihm enthaltene Zeichen genommen, bei einem numerischen Ausdruck das Zeichen, dessen ASCII-Code dem Wert des Ausdrucks entspricht. Alle Zeichen des entstehenden Strings sind identisch.

Beispiele:

```
STRING$ (5, "*")   ist gleichbedeutend mit "*****"
STRING$ (5, 42)    ist gleichbedeutend mit "*****"
STRING$ (12, "-")  ist gleichbedeutend mit "------------"
STRING$ (10, " ")  ist gleichbedeutend mit "          "
```

Im Gegensatz zu SPACE benutzt STRING$ den 7-bit ASCII-Code. Diesen können Sie direkt mit der Funktion ASC abrufen. Dieser Code darf Werte zwischen 1 und 255 annehmen. Allerdings sind eine ganze Reihe davon Steuerzeichen, die auf dem Bildschirm gar nicht darstellbar sind und somit für STRING$ auch kaum interessant sein dürften. So ist z.B. 13 der 7-bit ASCII Code für die RETURN-Taste.

Übrigens: ASCII hat keinesfalls etwas mit 'ASC II' zu tun, vielmehr steht es für 'American Standard Code for Information Interchange'. Soviel einmal zu einem weit verbreiteten Irrtum, dem Sie, verehrter Leser, nicht auch erliegen sollen.

Beispiel zu ASC:

```
PRINT ASC("*") liefert den Wert 42
STRING$ (3,42) ist gleichbedeutend mit "***"
```

STRING$ kann genau wie ein normaler String behandelt werden. Sie können es also zusammen mit PRINT genauso einsetzen wie zu Wertzuweisungen.

Beispiele:

```
PRINT STRING$ (4," ")     liefert den Ausdruck "    "
PRINT A;STRING$(10,".");B
F$= STRING$ (10,"?")      entspricht: F$= "??????????"
```

Die Länge des entstehenden Strings können Sie selbstverständlich auch berechnen lassen.

Beispiel:

```
100 DIM A$(20)
110 FOR I= 1 TO 20
120 A$(I)= STRING$ (I,"-")
130 NEXT I
140 MATRIX
```

Dieses Beispielprogramm erzeugt 20 Strings A(1) bis A(20), die alle aus dem Zeichen "-" bestehen, wobei jeder String genau eins länger ist als sein Vorgänger.

* INSTR (durchsuchstring,suchstring [,startposition])

Dieser Befehl gibt uns die Möglichkeit in die Hand, auf einfachste Weise festzustellen, ob ein String ('suchstring') in einem anderen String ('durchsuchstring') enthalten ist. Für das normale Basic gibt es eine ganze Reihe von Vorschlägen für Unterprogramme, die diese Funktion erfüllen. Da haben Sie es einfacher, Sie haben EXBASIC LEVEL II.

Das Ergebnis von INSTR ist die Position, ab welcher der Suchstring zum ersten Mal im Durchsuchstring vorkommt.

Beispiele:

```
PRINT INSTR ("BEISPIEL","EI")    führt zum Ausdruck 2
PRINT INSTR ("BEISPIEL","IE")    führt zum Ausdruck 6
PRINT INSTR ("BEISPIEL","XY")    führt zum Ausdruck 0
PRINT INSTR ("BEISPIEL","E")     führt zum Ausdruck 2
```

Wie das vorletzte Beispiel zeigt, ist das Ergebnis von INSTR null, falls der Suchstring gar nicht im Durchsuchstring enthalten ist. Ist einer der beiden Strings leer, so führt das zu einem ?ILLEGAL QUANTITY ERROR.

Zusammen mit INSTR kann noch eine Position angegeben werden, ab welcher der Durchsuchstring nach dem Suchstring durchsucht wird. Lassen Sie die Position weg, so wird automatisch dafür der Wert 1 angenommen, mithin also der gesamte String abgesucht.

Beispiele:

```
PRINT INSTR ("BEISPIEL","EI",2)  führt zum Ausdruck 2
PRINT INSTR ("BEISPIEL","EI",3)  führt zum Ausdruck 0
PRINT INSTR ("BEISPIEL","S",5)   führt zum Ausdruck 0
```

Beachten Sie, daß zwei Stringteile nur dann identisch sind, wenn sie auch in Groß-/und Kleinschreibung übereinstimmen.

Beispiel:

```
PRINT INSTR ("Beispiel","Ei")    führt zum Ausdruck 0
PRINT INSTR ("Beispiel","ei")    führt zum Ausdruck 2
```

Ein kleines Programm soll Ihnen das Ausprobieren von INSTR erleichtern:

```
100 INPUTFORM "Durchsuchstring"; D$
110 INPUTFORM "Suchstring"; S$
120 A= INSTR (D$,S$)
130 IF A=0 THEN 190
140 E= A + LEN (S$)
150 PRINT S$;" kommt in ";D$;" vor."
160 PRINT "Anfangsposition=";A
170 PRINT "Endposition=";E
180 STOP
190 PRINT "keine Übereinstimmung!"
```

* EVAL (string)

EVAL lehnt sich in gewisser Weise an die normale VAL-Funktion an. Während aber VAL nur Zahlen erkennt, arbeitet EVAL auch mit kompletten Funktionen in Form eines Stringausdrucks.

Beispiel:

```
VAL ("12*2")  liefert den Wert 12
EVAL ("12*2") liefert den Wert 24
```

EVAL erkennt selbstverständlich nicht nur die vier Grundrechenarten, sondern auch alle anderen numerischen Funktionen, wie Kreisfunktionen, logarithmische Funktionen, EXBASIC LEVEL II Funktionen etc.(EVAL in EVAL ist nicht erlaubt.). Damit eröffnet sich Ihnen eine Vielzahl von Anwendungsmöglichkeiten.

Beispiele:

```
PRINT EVAL ("4*(2+SIN(1.12))-52")
PRINT EVAL ("ATN(TAN(1))")
A= EVAL ("2.5/MIN(2,3,78,1)")
B= EVAL ("ODD(X)*SGN(X)+FRAC(X)")
```

EVAL und EXEC sind EXBASIC LEVEL II Befehle, die nicht im Direktmodus verwendbar sind, andernfalls kommt es zu einem ?ILLEGAL DIRECT ERROR.

Probieren Sie EVAL einmal mit dem folgenden kleinen Programm aus:

```
100 INPUTFORM "Funktion (Parameter X)"; F$
110 INPUT "Argument X=";X
120 PRINT EVAL (F$)
130 PRINT "F= Funktion, A= Argument ..ändern"
140 GET A$ : IF A$<> "F" AND A$<> "A" THEN.
150 IF A$= "F" RUN ELSE 110
160 END
```

Starten Sie das Programm mit RUN, und geben Sie dann eine beliebige Funktion mit dem Argument X ein, also z.B...

```
SIN (X) + COS (X)
```

Sie haben nun die Möglichkeit, sich für jeden Wert X das Ergebnis ausrechnen zu lassen. Wenn Sie eine neue Funktion eingeben wollen, drücken Sie 'F', sonst 'A' für 'Argument'. Wie das Programm zeigt, darf der String zu EVAL auch eine Stringvariable sein, im Beispiel ist es F$. Bedenken Sie aber, daß die Schachtelung von Anführungszeichen grundsätzlich nicht möglich ist.

Tja, so einfach ist ein solches Programm in EXBASIC LEVEL II. Machen Sie sich doch spasseshalber einmal ein paar Gedanken darüber, wie Sie dieses wirklich kurze Beispielprogramm im normalen Basic programmiert hätten.

* EXEC (string)

EXEC führt einen String, dessen Inhalt Basickommandos sind, wie im Programm stehend aus.

Beispiele:

```
EXEC ("LIST")       listet unter Programmkontrolle
EXEC (")IO")        initialisiert Laufwerk 0 (Floppy DOS 1.0)
EXEC ("]$1")        Inhaltsverzeichnis Laufwerk 1 (DOS 1.0)
EXEC ("←1:NAME") schreibt Programm "NAME" auf Laufwerk 1
```

Im Zusammenhang mit EXEC müssen Sie allerdings folgende Restriktionen hinnehmen: erstens darf der zu EXEC zugehörige String nur einen Basicbefehl enthalten und zweitens ist EXEC in EXEC nicht erlaubt. Diese Einschränkungen sind notwendig.

Übrigens: LIST braucht gar nicht in EXEC zu stehen, da in EXBASIC LEVEL II ein LIST-Befehl im Programm nicht zum Programmabbruch führt. Dagegen führen manche Befehle wie RENUM, AUTO etc. in jedem Fall zu einem Programmhalt, auch wenn sie in EXEC stehen.

Beispielprogramm zu EXEC:

```
100 INPUTFORM "Kommando"; K$
110 IF INSTR (K$,":") THEN 140
120 EXEC (K$)
130 GOTO 100
140 IF LEFT$ (K$,1)= "←" THEN 120
150 PRINT "Nur ein Kommando!"
160 RUN
```

In Zeile 100 können Sie einen beliebigen Basicbefehl eingeben, der in Zeile 110 daraufhin überprüft wird, ob er einen Doppelpunkt enthält. Falls dem nicht so ist, kommt es zur Ausführung mit EXEC (Zeile 120). Wenn der eingegebene Befehl einen Doppelpunkt enthält, so prüft das Programm, ob es sich um ein DOS 1.0 SAVE-Kurzkommando handelt (Abfrage in Zeile 140). Andernfalls wurde offensichtlich mehr als ein Basicbefehl eingegeben und die gesamte Eingabe wird zurückgewiesen (Zeile 150).

Beachten Sie zu EVAL und EXEC, daß im Commodore Basic die Schachtelung von Anführungszeichen grundsätzlich nicht möglich ist. So führt z.B. ...

```
100 EXEC ("PRINT"Hallo"")
```

..keineswegs zum Ausdruck 'Hallo', sondern zu einer Fehlermeldung. Wir können uns behelfen, indem wir 'Hallo' vorher einer Stringvariablen zuweisen:

```
99 A$="Hallo"
100 EXEC (PRINT A$)
```

* SEC [numerischer Ausdruck]

Wie haben Sie bisher einen Programmhalt für eine bestimmte Dauer programmiert? Wahrscheinlich so...

```
100 FOR I= 1 TO 2000 : NEXT I
```

..oder so ähnlich. Diese Warteschleifen haben nun endgültig ein Ende.

Nehmen wir einmal an, Sie möchten, daß Ihr Programm an einer Stelle exakt 5 Sekunden wartet. Das programmieren Sie...

```
100 SEC 5 :REM 5 Sek. Programmhalt
```

So einfach ist das in EXBASIC LEVEL II. Die Wartezeiten sind übrigens ziemlich exakt, prüfen Sie es einmal nach.

Der numerische Ausdruck zu SEC kann Werte zwischen 0 (sicher nicht sehr sinnvoll) und 255 annehmen, das ergibt Zeiten bis zu über 4 Minuten. Die Ausführung von SEC kann jederzeit durch Drücken der STOP-Taste abgebrochen werden.

* BEEP (tonhöhe (,tondauer)) / BEEP OFF

BEEP ermöglicht die Tonerzeugung mit dem Computer. Ohne weiteres ist dies allerdings nur bei Geräten der Serie 8001 machbar. Allen anderen Computern fehlt ein Lautsprecher. Ein anschlußfertiges Gerät zur Tonerzeugung für alle Serien inklusiv Interface, Verstärker und Lautsprecher können Sie unter der Bezeichnung MUSIKBOX beziehen beim Autor dieses Anleitungsbuches: Andreas Dripke, August-Bebel-Str. 38, 6200 Wiesbaden; genaue Informationen gegen Rückporto.

Zu BEEP können Sie die Tonhöhe und die Tondauer mit angeben, beide dürfen zwischen 0 und 255 sein. Für die Tonhöhe gilt dabei: Je größer die Zahl ist, desto tiefer ist der Ton. Wenn Sie sowohl Tonhöhe als auch Tondauer weglassen, so wird ein Ton mittlerer Höhe für eine Sekunde (Serien 2001/3001/4001) bzw. der charakteristische Beepton (Serie 8001) ausgesandt.

Beachten Sie, daß Sie zwar Tonhöhe und Tondauer zusammen weglassen dürfen, jedoch nicht nur die Tonhöhe. Wenn Sie also die Tondauer angeben wollen, dann müssen Sie auch eine Tonhöhe mitgeben. Dagegen ist die Angabe der Tonhöhe ohne Tondauerfestlegung erlaubt.

BEEP kann jederzeit durch Drücken der STOP-Taste unterbrochen werden. Dabei geht jedoch der Ton nicht aus. Sie können ihn aber mit BEEP OFF abschalten.

Fachleuten sei gesagt: BEEP benutzt den Ausgang CB 2 des USERPORTS.

BEEP ist zur Erzeugung von Signaltönen u.ä. gedacht. Musik dagegen können Sie mit BEEP praktisch nicht programmieren. Falls Sie hieran Interesse haben, so sollten Sie sich die genannte MUSIKBOX zulegen. Zusammen mit dieser erhalten Sie nämlich eine ausführliche Anleitung zur Musikprogrammierung. Die MUSIKBOX hat einen Ein-/Ausschalter und einen Lautstärkeregler eingebaut.

Für Musik ist übrigens auch der beim CBM 8001 eingebaute piezoelektrische Lautsprecher völlig ungeeignet. Dafür benötigen Sie schon einen kräftigen Verstärker, wie ihn die MUSIKBOX bietet.

Beispiele zu BEEP:

```
BEEP          Standard Beepton
BEEP 240      Ton 'C' für eine Sekunde
BEEP 240,1    Ton 'C' für eine Sekunde
BEEP 80,60    hoher Ton für eine Minute (60 Sekunden)
BEEP ,60      falsch! Es fehlt die Tonhöhenangabe!

BEEP OFF      schaltet den Ton nach Drücken von STOP aus
```

* ON ERROR GOTO [zeilennummer]

Bereits kennengelernt haben wir den Fehlermodus, der durch ON eingeschaltet wird. Erinnern wir uns: im Fehlermodus wird bei Auftreten eines Programmfehlers die entsprechende Zeile aufgelistet und der Cursor an der fehlerhaften Stelle positioniert.

Nun, das ist beim Austesten eines Programmes sicherlich sehr angenehm, was aber, wenn Ihr Programm erst einmal verkauft ist, wenn es sich beim Anwender befindet. Gerade bei kommerziellen Programmen kann ein Programmfehler zu sehr unangenehmen Folgen führen, angefangen bei einer völlig hilflosen Sekretärin bis hin zum Datenverlust.

Um so etwas zu vermeiden, gibt es in EXBASIC LEVEL II den Befehl ON ERROR GOTO. Denn daß ein von Ihnen geschriebenes Programm keinen Fehler mehr enthält, dessen können Sie sich nie sicher sein. Niemand kann das. Aber ist es nicht schon viel Wert, wenn Ihr Programm bei Auftreten eines Fehlers noch reagieren kann, wenn es den Fehler zwar erkennen, aber trotzdem noch weiter arbeiten kann?

In einer Fehlerbehandlungsroutine können Sie der Sekretärin eine deutsche Meldung zukommen lassen oder die aktuellen Daten auf Kassette oder Diskette abspeichern oder...

ON ERROR GOTO können Sie auch einsetzen, um einen durchaus voraussehbaren Fehler zu behandeln. Oft ist das viel einfacher, als den Fehler von vornherein auszuschließen. Ein typisches Beispiel für diesen Anwendungsfall ist die Division durch Null.

Mit ON ERROR GOTO [zeilennummer] können Sie festlegen, zu welcher Zeilennummer das Programm verzweigen soll, falls ein Fehler auftritt. ON ERROR GOTO muß festgelegt sein, bevor es zu einem Fehler kommt, andernfalls ist es unwirksam.

Beispiel:

```
10 ON ERROR GOTO 100
20 PRINT 1/0
```

Zeile 20 führt mit Sicherheit zu einem ?DIVISION BY ZERO ERROR. In diesem Fall wird zu Zeile 100 gesprungen, wo die Fehlerbehandlungsroutine liegen sollte.

Welche Art von Fehler auftrat, ist in der Variablen EC (error code) gespeichert, die Zeilennummer in EL (error line). Nachfolgend finden Sie eine Liste der Fehlercodes (EC):

00	MODUL	EXBASIC LEVEL II Meldung
01	BAD SUBSCRIPT	
02	DEVICE NOT PRESENT	
03	DIVISION BY ZERO	
04	FILE DATA	
05	FILE OPEN	
06	FILE NOT FOUND	
07	FILE NOT OPEN	
08	FORMAT	EXBASIC LEVEL II Meldung
09	FORMULA TOO COMPLEX	
10	ILLEGAL QUANTITY	
11	NEXT WITHOUT FOR	
12	NOT INPUT FILE	
13	NOT OUTPUT FILE	
14	OUT OF DATA	
15	OUT OF MEMORY	
16	OVERFLOW	
17	RESUME WITHOUT ERROR	EXBASIC LEVEL II Meldung
18	RETURN WITHOUT GOSUB	
19	REDIM'D ARRAY	
20	STRING TOO LONG	
21	SYNTAX	
22	TOO MANY FILES	
23	TYPE MISMATCH	
24	UNDEF'D STATEMENT	
25	UNDEF'D FUNCTION	
26	BAD DISK	nur Serien 4001 und 8001

Die Fehlermeldungen ?CAN'T CONTINUE ERROR und ?ILLEGAL DIRECT ERROR können nur im Direktmodus entstehen und somit mit ON ERROR GOTO nicht abgefangen werden.

Bedenken Sie bei der Programmierung in EXBASIC LEVEL II die Sonderstellung der beiden Variablen EC und EL. Sobald Sie ON ERROR GOTO verwenden, sollten Sie EC und EL ausschließlich zur Fehlererkennung heranziehen.
Fremde Programme können Sie mit Hilfe des Befehls FIND daraufhin untersuchen, ob sie die Variablen EC oder EL verwenden und diese nötigenfalls durch andere ersetzen.

Der ON ERROR GOTO Modus schließt Fehlermodus (ON) und STOP-Modus (STOP ON) aus. Schließlich kann bei Auftreten eines Fehlers entweder die Zeile gelistet oder zu einer bestimmten Programmzeile gesprungen oder laufend die STOP-Taste abgefragt werden. Alles auf einmal geht eben auch in EXBASIC LEVEL II nicht.

Um ON ERROR GOTO wieder auszuschalten, geben Sie ON ERROR GOTO 0 ein.

Die Fehlerbehandlungsroutine muß mit RESUME abgeschlossen werden. In dieser Routine selbst sollte kein Fehler auftreten.

* RESUME / RESUME NEXT / RESUME [zeilennummer]

RESUME mit seinen Variationen kennzeichnet das Ende einer Fehlerbehandlungsroutine, indem es spezifiziert, wo der normale Programmlauf wieder aufgenommen werden soll.

Bloßes RESUME ohne weitere Angaben bewirkt einen Rücksprung zu der Anweisung, welche den Fehler verursacht hat. Der Computer versucht, den Befehl erneut auszuführen. Gelingt das nicht, erfolgt ein erneuter Sprung in die Fehlerbehandlungsroutine. Diese Programmierung ist nur bei Eingabeanweisungen sinnvoll, wo die zunächst falsche Eingabe beim zweitem Mal korrigiert werden kann.

RESUME NEXT hat trotz des Wortes NEXT nichts mit FOR-NEXT-Schleifen zu tun. Vielmehr bewirkt es einen Rücksprung aus der Fehlerbehandlungsroutine auf den Befehl, der dem fehlerhaften Kommando unmittelbar folgt.

RESUME [zeilennummer] schließlich verursacht einen Sprung zur angegebenen Zeilennummer.

Beispielprogramm:

```
100 REM Quadratwurzelprogramm
200 REM ON ERROR einschalten
210 ON ERROR GOTO 900
300 REM Hauptprogramm
310 INPUT "Zahl"; X
320 PRINT "Wurzel:"; SQR(X)
330 GOTO 300
900 REM Fehlerbehandlungsroutine
910 PRINT SQR(-X); "*I"
920 RESUME 300
```

Normalerweise sind als Argumente für die Quadratwurzelfuntion SQR nur positive Werte einschließlich Null zugelassen. Starten Sie das Programm mit RUN und geben Sie auch negative Werte ein. Übrigens, in diesem Beispiel könnte in Zeile 920 statt 'RESUME 300' auch 'RESUME NEXT' stehen. Überlegen Sie einmal, warum.

Und noch ein Beispielprogramm:

```
100 REM Programm '1/X'
200 ON ERROR GOTO 900
300 INPUT "Zahl"; X
310 PRINT "Kehrwert="; 1/X
320 GOTO 300
900 REM Fehlerbehandlungsroutine
910 PRINT "Division durch Null!"
920 RESUME NEXT
```

Jetzt wollen wir uns noch einmal eine Fehlerbehandlungsroutine ansehen, die etwas allgemeiner zu verwenden ist als die bisher gezeigten:

```
1000 REM Fehlerbehandlungsroutine
1100 ON EC+1 RESTORE 1610,1620,1630...1870
1110 READ F$
1200 PRINT "Fehler: "; F$
1300 PRINT "Zeile:"; EL
1400 PRINT "Weitermachen (J/N)?"
1500 GET X$ : IF X$<>"J" AND X$<>"N" THEN.
1510 IF X$="J" RESUME NEXT ELSE END
1600 REM Liste der Fehlermeldungen
1610 DATA MODUL FEHLER
1620 DATA FALSCHER INDEX
1630 DATA GERÄT NICHT VORHANDEN
1640 DATA DIVISION DURCH NULL
1650 DATA FILE DATEN
1660 DATA FILE IST OFFEN
1670 DATA FILE NICHT GEFUNDEN
1680 DATA FILE NICHT OFFEN
1690 DATA FORMAT FEHLER
1700 DATA STRINGAUSDRUCK ZU KOMPLEX
1710 DATA BEREICHSÜBERSCHREITUNG
1720 DATA NEXT OHNE FOR
1730 DATA KEIN EINGABE FILE
1740 DATA KEIN AUSGABE FILE
1750 DATA KEINE DATEN MEHR FÜR READ VORHANDEN
1760 DATA BASICSPEICHER VOLL
1770 DATA ZAHL ZU GROSS
1780 DATA RESUME OHNE ERROR
1790 DATA RETURN OHNE GOSUB
1800 DATA FELD IST SCHON DIMENSIONIERT
1810 DATA STRING ZU LANG
1820 DATA SYNTAX FEHLER
1830 DATA ZU VIELE OFFENE FILES
1840 DATA FALSCHER VARIABLENTYP
1850 DATA ZEILENNUMMER NICHT VORHANDEN
1860 DATA UNBEKANNTE FUNKTION
1870 DATA FLOPPY FEHLER
```

Wenn der Computer an einen RESUME-Befehl stößt, ohne daß vorher ein Fehler aufgetreten ist, so erfolgt die Meldung ?RESUME WITHOUT ERROR.

* '

In EXBASIC LEVEL II dürfen Sie den Befehl REM einfach durch "'" abkürzen. Dadurch wird ein Programm in vielen Fällen übersichtlicher.

Beispiel:

```
100 REM Hier beginnt der Programmteil Eingabe
```

..ist in EXBASIC LEVEL II das gleiche wie...

```
100 'Hier beginnt der Programmteil Eingabe
```

* HARDCOPY

Mit dieser Funktion können Sie den Bildschirminhalt auf einem Drucker ausgeben. HARDCOPY funktioniert sowohl im Direktmodus als auch innerhalb eines Programmes. Damit können Sie also z.B. eine Graphik oder ein Formular auf dem Bildschirm zeichnen und später mit HARDCOPY zu Papier bringen.

Bei zu häufigem Wechsel RVS/OFF und umgekehrt innerhalb einer Bildschirmzeile mag es bei HARDCOPY zu Schwierigkeiten kommen, wenn die Pufferkapazität des Druckers nicht ausreichend bemessen ist. Daran läßt sich auch leider nichts ändern - es sei denn, Sie legen sich einen Drucker mit größerer Pufferkapazität zu.

HARDCOPY ist für die CBM-Matrixdrucker von Commodore gedacht. Bei Druckern anderer Art oder Herkunft arbeitet HARDCOPY u.U. nicht korrekt. Bei Nicht-Matrixdruckern entfällt z.B. im allgemeinen die Möglichkeit, Graphikzeichen auszugeben. Matrixdrucker fremder Anbieter sind manchmal im Interface nicht voll CBM-kompatibel.

* SWAP [variablenname 1],[variablenname 2]

Hiermit erhalten wir die Möglichkeit, die Werte zweier Variablen gleichen Typs ohne Zuhilfenahme einer dritten Variablen zu vertauschen.

Beispiele:

```
SWAP A,B     vertauscht die Werte von A und B
SWAP A%,B%   vertauscht die Werte von A% und B%
SWAP A$,B$   vertauscht die Werte von A$ und B$
```

Bei Variablen unterschiedlichen Typs erfolgt die Fehlermeldung ?TYPE MISMATCH ERROR.

Fehlermeldungen

EXBASIC LEVEL II hat alle Fehlermeldungen, die Sie schon vom normalen Basic her kennen. Wir haben deshalb darauf verzichtet, sie hier im einzelnen aufzuführen. Schlagen Sie diese bei Bedarf bitte in Ihrem Computerhandbuch nach.

Hier werden wir nur die neuen EXBASIC LEVEL II Fehlermeldungen besprechen sowie solche, die zwar im normalen Basic schon vorhanden sind, in EXBASIC LEVEL II jedoch einen leichten Bedeutungswandel durchgemacht haben. Letztere entfallen bei der Serie 8001 (s. unten).

* ?MODUL ERROR

Dieser Fehler kann ausschließlich im Zusammenhang mit SOFTMODUL-Funktionen oder -befehlen auftreten, also nur bei Verwendung von CALL. Die meisten SOFTMODULE bringen aber eigene Fehlermeldungen mit sich.

* ?FORMAT ERROR

Wenn bei PRINT USING die Anzahl der Formatzahlen nicht mit der Anzahl der vorgegebenen Zahlen übereinstimmt, so wird ein ?FORMAT ERROR gemeldet. Dies geschieht ebenfalls bei einem leeren Formatstring.

* ?RESUME WITHOUT ERROR

Ein ?RESUME WITHOUT ERROR wird ausgegeben, wenn das Programm an eine RESUME-Anweisung stößt, ohne daß vorher ein Fehler im ON ERROR GOTO Modus aufgetreten ist.

* ?FILE NOT FOUND ERROR (nicht Serie 8001)

Im Zusammenhang mit FAST TAPE LOAD zeigt ein ?FILE NOT FOUND ERROR, daß auf der Kassette zwar ein Programmkopf gefunden wurde, jedoch kein Programm.

* ?LOAD ERROR (nicht Serie 8001)

Während diese Fehlermeldung im normalen Basic nur bei LOAD auftreten kann, unterscheidet EXBASIC LEVEL II auch bei FAST TAPE VERIFY zwischen ?VERIFY ERROR und ?LOAD ERROR. Ersterer tritt auf, wenn das Programm auf Kassette ordentlich gelesen werden konnte, aber nicht mit dem im Basicspeicher befindlichen übereinstimmt. Falls die Programmaufzeichnung auf Kassette dagegen fehlerhaft ist, erfolgt ein ?LOAD ERROR.

Hinweise

Hier finden Sie eine Reihe von Besonderheiten, die Ihnen EXBASIC LEVEL II zur Verfügung stellt.

* WAIT-Befehl

EXBASIC LEVEL II erlaubt es Ihnen, im Gegensatz zum normalen Basic, auch einen WAIT-Befehl mit STOP zu unterbrechen. Damit kann eine falsche WAIT-Adresse dem Computer nicht mehr zum Verhängnis werden.

* Floppyfehler

Normalerweise leuchtet bei einem Floppyfehler lediglich die Fehlerlampe am Floppy rot auf. Bei Computern mit DOS 2.0 (Serien 4001/8001) können Sie die Fehlermeldung jederzeit mit...

```
PRINT DS$
```

..abrufen.

Bei Geräten mit DOS 1.0 (Serien 2001/3001) ist dies im normalen Basic nicht möglich, in EXBASIC LEVEL II geben Sie einfach...

```
]
```

..ein und drücken die RETURN-Taste. So einfach ist das.

* Restriktionen

Da sich EXBASIC LEVEL II an das Commodore Basic anlehnt (andernfalls müßten Sie fünf ROM-Bausteine auswechseln, was äußerst teuer wäre), mußten wir leider auch einige wenige Einschränkungen hinnehmen, die sich direkt aus dem Commodore Basic ergeben. Wir haben diese Restriktionen hier extern aufgeführt, da sie jeder Logik entbehren. Sie hängen mit einer schlechten Programmierung des Befehls NEXT im Commodore Basic zusammen.

1. Nach NEXT darf nicht direkt ELSE folgen. Sie können sich aber behelfen, indem Sie hinter NEXT einfach zwei Doppelpunkte schreiben. Diese haben weiter keinen Einfluß auf den Programmlauf, erlauben aber den Befehl ELSE direkt nach NEXT.

2. Wenn NEXT der letzte Befehl einer Zeile ist, so funktioniert in der direkt folgenden Zeile 'GOTO.' nicht korrekt. Nehmen Sie als Abhilfe 'GOTO zeilennummer'.

Kassettenrekorderkommandos

In die EXBASIC LEVEL II Versionen für die Serien 2001/3001 und 4001 ist ein komplettes FAST TAPE integriert. FAST TAPE bedeutet zunächst einmal, daß die Kassettenrekorderoperationen LOAD, SAVE und VERIFY mit fünffacher Geschwindigkeit ablaufen. Wenn Sie diese Funktionen mit normaler Geschwindigkeit ablaufen lassen wollen, so benutzen Sie bitte LOAD*, SAVE* oder VERIFY*.

Machen Sie sich also klar: Sobald EXBASIC LEVEL II aktiviert ist, arbeiten bei den genannten Serien die aufgeführten Befehle nicht mehr wie normal, sondern mit fünffacher Geschwindigkeit. Die Funktionen der normalen Kassettenrekorderbefehle werden durch Anhängen eines Asterisks '*' an den Befehl aufgerufen.

Beispiel:

LOAD	FAST TAPE LOAD Kommando
LOAD*	entspricht dem normalen LOAD Kommando

Als Benutzer des Kassettenrekorders werden Sie es sehr schnell zu schätzen wisssen, wenn Ihre Programme von nun an fünfmal so schnell abgespeichert oder geladen werden. Das bringt eine enorme Zeitersparnis mit sich.

Des weiteren bietet EXBASIC LEVEL II FAST TAPE noch eine Reihe anderer Annehmlichkeiten. So dürfen Programmnamen grundsätzlich bis zu 30 Zeichen lang sein, normal sind es höchstens 16 Zeichen. Wir werden bei den einzelnen Befehlen auf die jeweiligen Besonderheiten zu sprechen kommen.
Außerdem stellt FAST TAPE zwei neue Befehle für den Kassettenrekorder zur Verfügung: MERGE und MOD. Da MERGE auch zusammen mit einer Floppystation funktioniert, finden Sie es auch unter 'Hilfsfunktionen' aufgeführt.

* LOAD ("name") (,startadresse)

LOAD ohne Angabe einer Startadresse wirkt wie bisher, aber eben mit fünffacher Geschwindigkeit. Voraussetzung dafür ist natürlich, daß Sie das betreffende Programm mit FAST TAPE SAVE auf Kassette geschrieben haben.

Am Anfang werden Sie noch viele Programme in normaler langsamer Aufzeichnung auf Kassette gespeichert haben. Diese können Sie mit LOAD* einlesen. Wir empfehlen Ihnen aber, nach und nach Ihr gesamtes Programmarchiv auf FAST TAPE Aufzeichnung umzustellen. Das kostet zwar einmal viel Zeit, es spart aber hinterher ein Vielfaches an Zeit davon.

Wenn Sie zu LOAD eine Startadresse angeben, so wird die auf Band stehende Startadresse ignoriert und das Programm an die im LOAD-Befel angegebene Adresse geladen. Das ist im allgemeinen für Basicprogramme nicht sinnvoll, dafür aber bei Maschinenprogrammen umso nützlicher.

Da EXBASIC LEVEL II FAST TAPE eine durch Komma getrennte Zahl hinter dem Programmnamen als Startadresse interpretiert, kann...

LOAD "programmname",8

..nicht mehr zum Laden von Floppydisk benutzt werden. Die Sekundäradresse der Floppystation hat die Nummer 8.

Nehmen Sie vielmehr in Zukunft dafür den Befehl...

LOAD* "programmname",8

Dies lädt das Programm "programmname" von Floppydisk. Einfacher ist natürlich bei den Serie 2001/3001 und 4001 die DOS SUPPORT Befehlsabkürzung...

/programmname

oder z.B....

/0:programmname

..um ein bestimmtes Laufwerk (im Beispiel 0) zu spezifizieren.

Wenn Sie einen Computer der Serie 4001 benutzen, so steht Ihnen zum Laden eines Programmes auch der DOS 2.0 Befehl DLOAD zur Verfügung.

Beispiele:

```
DLOAD "DEMO"        lädt Programm "DEMO" von Laufwerk 0
DLOAD "DEMO",D0     lädt Programm "DEMO" von Laufwerk 0
DLOAD "DEMO",D1     lädt Programm "DEMO" von Laufwerk 1
```

Bei der Serie 8001 ist in EXBASIC LEVEL II gar kein FAST TAPE eingebaut, weswegen dort solche Verwechslungen von Start- und Sekundäradresse bei LOAD gar nicht möglich sind.

Beachten Sie zu FAST TAPE LOAD, daß der Ladevorgang nicht wie üblich durch Drücken der STOP-Taste am Computer abgebrochen werden kann. Drücken Sie zu diesem Zweck einfach die STOP-Taste am Kassettenrekorder. Es erscheint dann ein ?LOAD ERROR und der Ladevorgang wird abgebrochen.

Solange FAST TAPE LOAD noch kein Programm auf Kassette gefunden hat, können Sie den Suchvorgang durch Drücken der STOP-Taste am Computer abbrechen. Danach, wie gesagt, nur noch mit STOP am Rekorder.

In EXBASIC LEVEL II FAST TAPE signalisiert Ihnen ein ?FILE NOT FOUND ERROR im Zusammenhang mit LOAD, daß zwar ein Programmkopf gefunden wurde, aber kein Programm.

* SAVE ("name") (,startadresse,endadresse)

Der EXBASIC LEVEL II FAST TAPE Befehl SAVE arbeitet, wie beschrieben, mit fünffacher Geschwindigkeit gegenüber dem SAVE-Befehl des normalen Basics. Solcherart geschriebene Programme können Sie nur mit FAST TAPE LOAD wieder laden.
Die normale SAVE-Funktion (einfache Geschwindigkeit) steht Ihnen auch weiterhin in Form des Befehls SAVE* zur Verfügung.

Wenn Sie zu SAVE keine Start- und Endadresse angeben, so wirkt es ganz normal auf den Basic Speicherbereich. Wenn Sie also nicht mit Maschinenprogrammen arbeiten, brauchen Sie sich darum gar nicht zu kümmern. Durch die Angaben von Start- und Endadresse können Sie einen beliebigen Speicherbereich auf Kassette aufzeichnen. Dadurch ist das Abspeichern von Maschinenprogrammen sehr einfach. Um z.B. den zweiten Kassettenpuffer zu speichern, geben Sie...

```
SAVE "ZWEITER KASSETTENPUFFER", 826, 1017
```

..ein. Bedenken Sie, daß in EXBASIC LEVEL II FAST TAPE der Name bis zu 30 Zeichen lang sein darf. Sie können sich also die Einführung von unsinnigen Abkürzungen für Ihre Programmnamen von Anfang an ersparen.

Übrigens kann mit FAST TAPE SAVE auch ein Speicherbereich oberhalb von 32768 (8000 hex) auf Kassette geschrieben werden, im Gegensatz zu den sonst üblichen Vefahren.

FAST TAPE SAVE beginnt sofort nach Drücken der RETURN-Taste mit der Programmaufzeichnung. Ein evtl. Vorspann der Kassette muß also vorgespult werden, sonst ist das Programm später nicht mehr einzulesen.
Während des FAST TAPE SAVE ist kein Abbruch durch die STOP-Taste möglich, auch nicht durch Drücken der STOP-Taste am Kassettenrekorder. Im diesem Fall hilft nur noch ein wenig Geduld.

* VERIFY

Mit dem EXBASIC LEVEL II FAST TAPE Befehl VERIFY können Sie ein auf Kassette gespeichertes Programm daraufhin überprüfen, ob es mit dem im Speicher befindlichen übereinstimmt. Für Programme, die nur mit einfacher Geschwindigkeit aufgezeichnet sind, verwenden Sie VERIFY*.

Wenn die Programme im Speicher und auf Kassette übereinstimmen, erfolgt wie üblich die Meldung OK, bei Ungleichheit ein ?VERIFY ERROR. Falls die Bandaufzeichnung fehlerhaft ist, erscheint, im Gegensatz zum normalen Basic, die Fehlermeldung ?LOAD ERROR. Durch diese Unterscheidung können Sie leicht überprüfen, ob ein Programm auf Kassette nur nicht mit dem im Speicher stehenden identisch ist oder ob die ganze Bandaufzeichnung unbrauchbar ist.

* MOD ("name") (,startadresse) (,endadresse) (,runadresse)

Dieser nur in EXBASIC LEVEL II FAST TAPE verfügbare Befehl erzeugt ein sich beim Einlesen selbst startendes Programm auf Kassette. Das Programm kann mit normalen LOAD von jedem anderen Computer auch ohne EXBASIC LEVEL II eingelesen werden, trotzdem lädt es fünfmal schneller als normal. Na, ist das nicht fantastisch?

Wenn Sie Start-, End- und Runadresse weglassen, bezieht sich MOD auf den normalen Basic Speicherbereich. Durch Angabe der Adresswerte können Sie Maschinenprogramme mit MOD abspeichern. Sobald ein solcherart aufgezeichnetes Maschinenprogramm eingelesen ist, beginnt es an der Runadresse mit der Abarbeitung.

Mit MOD können wir auf sehr einfache Weise Programme auf Kassette aufzeichnen, die weder kopier- noch LISTbar sind. Dazu muß der erste (!) Befehl unseres Programmes wie folgt lauten:

```
DOKE 144, 58929 (Serie 2001/3001)
DOKE 144, 58456 (Serie 4001)
```

Dadurch wird nämlich die STOP-Taste ausgeschaltet. Sobald das Programm also eingeladen ist, fängt es an zu laufen und ist auch durch STOP nicht mehr abbrechbar.

Sie sollten in einem solchen geschützten Programm anstelle des normalen INPUT-Befehls INPUTLINE oder INPUTFORM verwenden. Bei INPUT ist nämlich doch noch ein Herauskommen aus dem Programm durch bloßes Drücken der RETURN-Taste ohne weitere Eingaben möglich. INPUTLINE oder INPUTFORM unterbindet dies.

* MERGE ("name")

MERGE erlaubt Ihnen, zwei Programme zusammenzufügen, die beliebige, aber unterschiedliche Zeilennummern haben. MERGE wirkt also ähnlich wie LOAD, indem es ein Programm lädt, aber ohne dabei das schon vorhandene Programm zu zerstören. Vielmehr werden die neu hinzukommenden Zeilennummern an den richtigen Stellen hinzugefügt.

MERGE funktioniert auf einem Kassettenrekorder nur zusammen mit FAST TAPE Programmen, nicht mit solchen, die mit einfacher Geschwindigkeit aufgezeichnet wurden.

MERGE arbeitet auch zusammen mit einer Floppystation (MERGE*, auch Serie 8001) und ist deshalb ausführlich unter 'Hilfsfunktionen 13' beschrieben. Schauen Sie also bitte dort noch einmal nach.

Floppykurzbefehle

Die Commodore Computer der Serien 2001 und 3001 besitzen zur Floppysteuerung das Disk Operating System DOS 1.0. Wenn Sie damit arbeiten, dann wissen Sie auch, daß dieses nicht gerade sehr komfortabel ist. In EXBASIC LEVEL II haben wir für die Serien 2001, 3001 und 4001 das DOS SUPPORT eingebaut, wodurch die Floppysteuerung wesentlich vereinfacht wird.

* >disk befehl

Das Zeichen '>' mit nachfolgendem Disk-Befehl ersetzt die Befehlsfolge...

```
OPEN 15,8,15
PRINT#15, "disk befehl"
CLOSE 15
```

Als Disk-Befehl können Sie jedes DOS 1.0 Kommando einsetzen.

Beispiele:

```
>I1          Laufwerk 1 wird initialisiert
>$0          Inhaltsverzeichnis von Laufwerk 0
>S0:NAME     löscht Programm NAME von Laufwerk 0
>N1:KOPF,XY  formatiert Laufwerk 1

>C1:NAME NEU=0:NAME ALT kopiert 0 nach 1 (Namensänderung)
>R0:NAME NEU=0:NAME ALT Umbenennen auf Laufwerk 0
```

Bitte ziehen Sie Ihr Floppyhandbuch zum genaueren Studium des DOS 1.0 Befehlsvorrats zu Rate.

* >

Wenn Sie dem Zeichen '>' keinen Disk-Befehl folgen lassen, sondern nur RETURN drücken, so wird der Floppystatus bzw. die Floppyfehlermeldung ausgedruckt.

Beispiel:

Sie geben...

```
>
```

..ein und drücken RETURN und es erscheint...

```
00,OK,00,00
```

..als Floppystatus. Das zeigt Ihnen, daß die letzte Diskettenoperation ordnungsgemäß abgeschlossen wurde. Einen genauen Überblick über alle Meldungen finden Sie in Ihrem Floppyhandbuch.

* /[name]

Das Zeichen '/' stellt im EXBASIC LEVEL II DOS SUPPORT eine Abkürzung für LOAD von Diskette dar.

Beispiel:

```
/DEMO        lädt das Programm mit Namen "DEMO"
```

Der Name muß ohne Anführungszeichen angegeben werden. Um auf einem bestimmten Laufwerk suchen zu lassen, stellen Sie die Laufwerksnummer durch Doppelpunkt getrennt dem Programmnamen voran.

Beispiele:

```
/0:DEMO      lädt "DEMO" von Laufwerk 0
/1:DEMO      lädt "DEMO" von Laufwerk 1
```

* ↑[name]

Alles über '/' Geschriebene gilt ohne Einschränkung auch für '↑'. Der einzige Unterschied ist, daß ein mit '↑' geladenes Programm nach dem Einladen automatisch zu arbeiten anfängt. Sie ersparen sich also auf diese Weise das Eintippen des Wortes RUN.

Beispiele:

```
↑AUTORUN     lädt "AUTORUN" und startet es automatisch
↑0:AUTORUN   lädt "AUTORUN" von Laufwerk 0
↑1:DEMO      lädt "DEMO" von Laufwerk 1
```

* ←[0:name] oder ←[1:name]

Der Pfeil nach links ist in EXBASIC LEVEL II DOS SUPPORT eine Abkürzung für SAVE auf Diskette. Beachten Sie unbedingt, daß im Gegensatz zu den LOAD-Abkürzungen bei SAVE die Laufwerksnummer (0 oder 1) mit angegeben werden sollte.

Beispiele:

```
←0:BEISPIEL speichert Programm "BEISPIEL" auf Laufwerk 0
←1:BEISPIEL speichert Programm "BEISPIEL" auf Laufwerk 1
```

* MERGE* "name"

MERGE* erlaubt Ihnen, zwei Programme zusammenzufügen, die beliebige, aber unterschiedliche Zeilennummern haben. MERGE* stellt eine Hilfsfunktion dar und Sie finden deshalb eine genaue Beschreibung dieses Kommandos auch dort.

Bildschirmsonderbefehle

In EXBASIC LEVEL II für die Serie 8001 haben wir ein SCREEN SUPPORT integriert, das Ihnen eine Reihe von äußerst leistungsfähigen Sonderbefehlen für die Bildschirmsteuerung des Computers zur Verfügung stellt. Alle SCREEN SUPPORT Befehle können Sie sowohl im Direktmodus benutzen als auch programmieren. Sinnvoll sind sie aber eigentlich nur innerhalb eines Programmes.

* SCREEN (spalte lo,zeile lo, spalte ru,zeile ru)

Zu den Abkürzungen: 'lo' steht für "links oben", 'ru' für "rechts unten".

SCREEN mit Angabe zweier Punkte definiert den durch die beiden Punkte festgelegten Bildschirmausschnitt als Teilbildschirm (Bildschirmfenster). Alle Bildschirmfunktionen wirken nach einer solchen Definition nur noch auf diesen Bildschirmausschnitt, der restliche Schirm bleibt unberührt. Damit können Sie z.B. einen bestimmten Teil des Bildschirms löschen, während alles andere stehenbleibt.

Welche beiden Punkte Sie festlegen müssen, finden Sie noch einmal auf folgender Skizze veranschaulicht. A ist der Punkt links oben, B derjenige rechts unten. Beachten Sie, daß die Zeilennumerierung von 0 bis 24, die Spaltennumerierung von 0 bis 79 geht.

```
A--------------------
I  durch SCREEN de-  I
I  finierter Bild-   I
I  schirmausschnitt  I
--------------------B
```

Beispiel:

```
100 SCREEN 40,0,79,24 : PRINT "Bildschirm löschen"
```

definiert den rechten Bildschirmteil als Bildschirmfenster und löscht diesen. Die linke Bildschirmhälfte bleibt unberührt.
In vielen Fällen ist es auch sehr hilfreich, Systemzeilen am oberen und unteren Ende des Bildschirms vor dem Bildschirmrollen zu schützen, indem man den Teil des Bildschirms, in dem geschrieben werden soll, als Fenster definiert. Damit entfällt die Notwendigkeit, den Inhalt der Systemzeilen (oder auch System-Rand-Spalten) nach jedem Rollen oder Löschen des Bildschirms zu restaurieren.

Um auf die normale Bildschirmgröße von 25 x 80 zurückzuschalten, geben Sie SCREEN ohne weitere Parameter an:

```
110 SCREEN  stellt den Ursprungszustand her
```

* DELLINE (numerischer ausdruck)

DELLINE steht für 'DELETE LINE'. Die Wirkung ist wie folgt: Alle Zeilen unterhalb der Zeile, in welcher der Cursor steht, werden um eine Zeile hochgezogen. Dadurch wird die Cursorzeile selbst mit der folgenden Zeile überschrieben. Die unterste Zeile des Bildschirms wird mit Leerzeichen gelöscht.

Beispiel:

```
100 PRINT "Bildschirm löschen";
110 FOR I= 49 TO 53
120 PRINT STRING$ (50,I)
130 NEXT I
140 PRINT@ 80; 'zweite Bildschirmzeile
150 DELLINE
```

..führt zum Ausdruck von...

```
11111111111111111111111111111111111111111111111111
33333333333333333333333333333333333333333333333333
44444444444444444444444444444444444444444444444444
55555555555555555555555555555555555555555555555555
```

Die Zeile mit den Zweien wird durch DELLINE gelöscht.

Zu DELLINE kann ein numerischer Ausdruck mit angegeben werden. Die Operation DELLINE wird entsprechend oft ausgeführt.

```
DELLINE 10
```

..bedeutet also: lösche 10 Zeilen in der Art, wie oben angegeben. Der numerische Ausdruck kann jeden Wert zwischen 1 und 255 haben. Allerdings sind Zahlen größer als 25 in diesem Zusammenhang sinnlos.

* INSTLINE (numerischer ausdruck)

INSTLINE ist als 'INSERT LINE' zu verstehen. Während die Taste INST nur Platz für jeweils ein Zeichen schafft, können Sie mit INSTLINE ganze Freizeilen schaffen.

Im Detail wirkt INSTLINE wie folgt: Alle Zeilen ab einschließlich der Zeile, in welcher der Cursor steht, werden um eine Zeile nach unten geschoben. Dadurch wird die letzte Zeile nach unten aus dem Bildschirm herausgeschoben. Die Cursorzeile wird mit Leerzeichen gelöscht.
Die unterste Bildschirmzeile geht dabei verloren, wenn sie nicht durch SCREEN entsprechend geschützt ist.

Wenn zu INSTLINE ein numerischer Ausdruck angegeben ist, so wird die Operation INSTLINE entsprechend oft ausgeführt. Als Wert kommt wie bei DELLINE jede Zahl zwischen 1 und 255 in Betracht, Werte größer 25 sind aber im allgemeinen sinnlos.

* ENDLINE

ENDLINE löscht alle Zeichen rechts der Cursorposition einschließlich bis zum Zeilenende. Der Cursor steht nach ENDLINE an derselben Stelle wie vorher.

Wenn der Cursor am Anfang der Zeile steht, können Sie mit ENDLINE also die ganze Zeile löschen. Beachten Sie dabei aber die Unterschiede zu DELLINE und INSTLINE.
DELLINE löscht zwar wie ENDLINE die Cursorzeile, zieht aber alle nachfolgende Zeilen um eins nach oben. Wo vorher die Cursorzeile stand, befindet sich nach DELLINE also die nächstfolgende Zeile.
INSTLINE schafft zwar wie ENDLINE eine Leerzeile, wo vorher die Cursorzeile stand, schiebt aber alle nachfolgenden Zeilen einschließlich der Cursorzeile um eins nach unten. Gelöscht wird dabei im Grunde gar nicht die Cursorzeile, die nur verschoben wird, sondern die letzte Zeile des Bildschirms, die bei INSTLINE verloren geht.

Unabhängig davon wirken DELLINE und INSTLINE in jedem Fall auf ganze Zeilen, während ENDLINE eine Zeile erst ab der Cursorposition löscht. Außerdem wirkt ENDLINE in jedem Fall nur innerhalb einer Zeile, während Sie bei DELLINE und INSTLINE zusätzlich angeben können, wie oft die betreffende Operation ausgeführt werden soll.

Beispiel zu ENDLINE:

```
100 PRINT@ 160, "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
110 PRINT@ 160;
120 PRINT TAB(15);
130 ENDLINE
```

Dieses Beispielprogramm druckt erst das Alphabet aus und löscht es danach ab dem 15. Buchstaben wieder.

Ein anderes Beispiel:

```
100 PRINT "Eingabe"; : ENDLINE : INPUTLINE X$
```

ENDLINE löscht den Rest der Zeile, damit diese für die nachfolgende Eingabe mit INPUTLINE frei ist.
Natürlich könnten Sie statt INPUTLINE auch INPUTFORM nehmen, dann werden nämlich sowieso so viele Zeichen gelöscht, wie Sie bei INPUTFORM spezifiziert haben.

* BEGINLINE

BEGINLINE ist der zu ENDLINE zugehörige Befehl. BEGINLINE löscht alle Zeichen links der Cursorposition ausschließlich, beginnend beim Zeilenanfang.

Nehmen wir wieder unser kurzes Beispielprogramm mit dem Alphabet:

```
100 PRINT@ 160, "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
110 PRINT@ 160;
120 PRINT TAB(15);
130 BEGINLINE
```

Diesmal werden alle Buchstaben bis zum 15. Zeichen gelöscht. Dieser selbst bleibt unberührt.

* SCREEN UP / DOWN (numerischer ausdruck)

Die SCREEN SUPPORT Befehle SCREEN UP und SCREEN DOWN ermöglichen es Ihnen, den Bildschirminhalt nach Belieben nach oben oder unten zu rollen.

Mit SCREEN UP werden alle Zeilen des Bildschirms um eine Zeile nach oben gerollt. Die oberste Zeile rollt am oberen Bildschirmrand heraus, die unterste Zeile wird durch Leerzeichen ersetzt.
Bei SCREEN DOWN werden alle Zeilen des Bildschirms um eine Zeile nach unten gerollt. Die unterste Zeile rollt dabei am unteren Bildschirmrand heraus, die oberste Zeile wird durch Leerzeichen ersetzt.

Sowohl SCREEN UP als auch SCREEN DOWN beeinflussen ausschließlich das mit SCREEN definierte Bildschirmfenster. Der Rest des Bildschirms bleibt unangetastet. Das gilt natürlich nur, falls Sie überhaupt ein solches Fenster definiert haben. Andernfalls nimmt der Computer an, daß quasi der gesamte Bildschirm auch das Fenster ist, auf das SCREEN UP und SCREEN DOWN wirken soll.

In dem numerischen Ausdruck können Sie spezifizieren, wie oft die Funktion SCREEN UP bzw. SCREEN DOWN ausgeführt werden soll. Der Wert des Ausdrucks kann zwischen 1 und 255 liegen. Falls Sie keinen Wert angeben, wird als Ersatzwert 1 angenommen, es wird also um eine Zeile gerollt.

Beispiel:

```
SCREEN UP 10   rollt 10 Zeilen nach oben
```

Wenn Sie während des "Rollvorganges" die Taste ':' drücken, so friert dies den Bildschirm ein, der Computer hält in der Programmausführung inne. Mit 'Pfeil nach links' ist eine Fortsetzung ("auftauen") möglich. Dies gilt auch bei DELLINE und INSTLINE.

Machen wir uns noch einmal die Unterschiede zwischen den einzelnen SCREEN SUPPORT Kommandos zur Bildschirmsteuerung klar. Falls mit SCREEN ein Bildschirmfenster definiert wurde, beziehen sich die Angaben oberste bzw. unterste Zeile in der folgenden Tabelle immer auf dieses Fenster.

Befehl	Cursorzeile	oberste Zeile	unterste Zeile
INSTLINE	Leerzeile	unbeeinflußt	geht verloren
SCREEN DOWN	nächstobere	Leerzeile	geht verloren
DELLINE	geht verloren	unbeeinflußt	Leerzeile
SCREEN UP	nächstuntere	geht verloren	Leerzeile
ENDLINE (Anfang)	Leerzeile	unbeeinflußt	unbeeinflußt

Die Angaben für ENDLINE gelten nur, wenn der Cursor bei Ausführung von ENDLINE am Zeilenanfang der Cursorzeile positioniert ist.

BEGINLINE und ENDLINE lassen prinzipiell die oberste und unterste Bildschirmzeile unbeeinflußt. Die Cursorzeile wird bis zu bzw. ab der Cursorposition durch Leerzeichen ersetzt.

SCREEN* / SCREEN

Viele haben sich darum bemüht, wir haben es geschafft: den 80 Zeichen Bildschirm der Serie 8001 voll kompatibel zum 40 Zeichen Bildschirm der anderen Serien zu machen!

Für die Serien 2001, 3001 und 4001 gibt es ein reichhaltiges Angebot an Software. Die meisten der Programme funktionieren aber nicht auf Computern der Serie 8001 wegen der unterschiedlichen Bildschirmgröße. Nun gibt es ja auch im normalen Basic der Serie 8001 die Möglichkeit, ein Bildschirmfenster zu definieren. Es bietet sich also an, ein 40 x 25 Fenster in der Mitte des Bildschirms festzulegen, um so auch Programme ablaufen zu lassen, die für diese Bildschirmgröße geschrieben sind. Dieser Trick hat aber einen entscheidenden Nachteil: PEEK und POKE Befehle, die sich auf den Bildschirm beziehen, werden durch die Bildschirmfensterdefinition nicht umgestellt. Viele Programme, und gerade die besten, benutzen aber PEEK und POKE. Wenn ein solches Programm nun auf einem Computer der Serie 8001 abläuft, kommt es zum "Bildschirmchaos".

Mit SCREEN* verleihen Sie der Serie 8001 einen kleinen 40 x 25 Zeichen Bildschirm, in dem alle PRINT-, PEEK- und POKE-Befehle korrekt funktionieren. Die niedrigste Adresse des kleinen Bildschirms ist 32768, die höchste Adresse 33767, genau wie bei den Serien 2001, 3001 und 4001.

SCREEN* ist im Grunde der einzige SCREEN SUPPORT Befehl, der auch im Direktmodus sinnvoll ist. Aber natürlich ist SCREEN* auch programmierbar. Es ist eine gute Idee, alle Programme, die einen Bildschirm mit 40 Zeichen Breite brauchen, als einen der ersten Programmbefehle mit SCREEN* auszustatten, soweit dies möglich ist.

Mit dem Befehl SCREEN schalten Sie den Bildschirm wieder auf die normale Größe von 80 x 25 Zeichen um. Gleichzeitig werden damit auch alle Bildschirmfensterdefinitionen aufgehoben.

Wenn Sie mit SCREEN* den kleinen 40 x 25 Zeichen Bildschirm eingestellt haben, sollten Sie es vermeiden, zweimal direkt hintereinander 'HOME' zu drücken. Auch in den Programmen sollten Sie zweimal direkt hintereinander 'HOME' vermeiden. Mit zweimal 'HOME' wird nämlich im Commodore Basic für die Serie 8001 die Bildschirmfensterdefinition aufgehoben, was bei kleinem Bildschirm zu recht merkwürdigen Effekten führt.

Ebenfalls aufgehoben wird die Bildschirmfensterdefinition durch die Befehle LETTER und LETTER OFF.
Verwenden Sie deswegen...

```
POKE 59468,14   anstelle von LETTER
POKE 59468,12   anstelle von LETTER OFF
```

Bei kleinem Bildschirm arbeitet TAB u.U. nicht korrekt. Ersetzen Sie in diesem Fall TAB durch SPC oder STRING$(..,32). Zu dem Effekt kommt es durch die Doppelzeilenstruktur der Serien 2001, 3001, 4001, die der 8001 nicht kennt.

Der Befehl SCREEN* ist dazu gedacht, Programme, die für einen 40 x 25 Zeichen Bildschirm geschrieben sind, auf einem Computer der Serie 8001 ablaufen lassen zu können. Die Betonung liegt dabei auf "ablaufen". Probieren Sie nicht, ein Programm auf diesem kleinen Bildschirm einzugeben oder zu korrigieren. Der Grund dafür ist, daß die Serien 2001, 3001 und 4001 Doppelzeilen kennen, also Programmzeilen, die sich über zwei Bildschirmzeilen erstrecken, während die Serie 8001 grundsätzlich nur Einfachzeilen verarbeitet. Wenn Sie nun auf dem kleinen Bildschirm Programmzeilen eingeben wollen, so mag das zu recht kuriosen Effekten führen.
Wenn Sie ein Programm eingeben oder korrigieren möchten, schalten Sie grundsätzlich mit SCREEN auf den großen Bildschirm. Zum Austesten können Sie ja wieder mit SCREEN* umschalten.

Assembler/Disassembler

In EXBASIC LEVEL II für die Serie 8001 bewirkt der Befehl GO nicht wie bei den anderen Serien den Einsprung in TIM, sondern in EXMON. EXMON ist ein Maschinensprachemonitor für die Commodore Computer der Serie 8001, der alle TIM-Befehle und zusätzlich Assembler- und Disassemblerkommandos kennt.

Gestartet wird EXMON, wie gesagt, mit GO. Danach stehen Ihnen die normalen Basic und EXBASIC LEVEL II Befehle nicht mehr zur Verfügung, sondern nur noch die EXMON Kommandos.

EXMON ist für Sie im Grunde nur dann interessant, wenn Sie sich, wenigstens etwas, in der 6502 Assemblersprache auskennen. Immerhin können Sie EXMON auch dazu benutzen, Assemblerlistings, die hin und wieder veröffentlich werden, einzutippen.

Noch ein paar Hinweise zur hier verwendeten Befehlssyntax. EXMON hat einen veränderten Bildschirmeditor gegenüber dem normalen EXBASIC LEVEL II. In EXMON melden sich alle Kommandozeilen mit einem Punkt. Wir haben dies bei der folgenden Darstellung der EXMON Befehle berücksichtigt.
Außerdem wollen wir die Vereinbarung treffen, daß Kleinbuchstaben bei der Befehlsdarstellung für beliebige hexadezimale Zahlen stehen. So bedeutet...

xxxx yyyy

..etwa, daß hier zwei Hexzahlen angegeben werden müssen, also z.B...

033A 034F

Alle EXMON Befehlsworte bestehen aus lediglich einem einzigen Buchstaben.

. A xxxx op-code mit adresse

ASSEMBLER - Assemblierung des OP-Code mit Adresse an die mit xxxx angegebene Speicherstelle.

EXMON arbeitet grundsätzlich hexadezimal, alle Adressen müssen also ebenfalls als Hexzahl angegeben werden. Die Syntax für den Assembler ist gleich mit der des Disassemblers. Bei Zweifeln irgendwelcher Art können Sie sich also einfach ein Maschinenprogramm disassemblieren lassen und die Syntax dort abschauen.

Beispiel:

```
.A 033A LDA #$22
.A 033C STA $8000
.A 033F BRK
```

Wenn Sie einmal mit A den Assembler angesprochen haben, gibt er nachfolgende Adressen automatisch aus. Um aus dem A-Modus wieder herauszukommen, drücken Sie anstelle der Eingabe eines OP-Codes nur die RETURN-Taste.

. C xxxx yyyy

CALCULATE BRANCH - Berechnung des Offsets für einen relativen Sprung von xxxx zu yyyy.

Mit C können Sie die Offsetadresse eines relativen Verzweigungsbefehls (BEQ, BNE, BCC, BCS, BMI, BPL, BVC, BVS) berechnen.

.C 033C 033A

..rechnet den Offset eines Sprunges von 033C nach 033A aus. Der Assembler verlangt aber grundsätzlich absolute Sprungadressen.

. D xxxx

DISASSEMBLER - Disassemblierung der Befehle von xxxx bis der Bildschirm voll ist.

.D 033A

..disassembliert ab 033A solange, bis der Bildschirm gefüllt ist. Die Bytes hinter der Adresse können modifiziert werden. Dazu gehen Sie einfach mit dem Cursor an die entsprechende Stelle und überschreiben sie. Sobald Sie RETURN drücken, wird die Änderung wirksam und der gesamte Bereich neu ausgegeben. Sie haben somit sofort die Gewißheit darüber, was Ihre Eingabe bewirkt hat. OP-Codes dürfen an dieser Stelle nicht eingegeben werden. Nehmen Sie dazu den Befehl A zu Hilfe.

Um nachfolgende Adressen zu disassemblieren, geben Sie einfach D und RETURN ein. Der Cursor ist nach der Ausgabe einer Disassemblerseite so positioniert, daß D (RETURN) die Ausgabe der nächsten Bildschirmseite bewirkt.

. G xxxx

GO - Ausführung eines Maschinenprogramms ab Adresse xxxx solange, bis der Befehl BRK (00) auftritt.

Ist keine Adresse angegeben, wird als Ersatz der Programmzähler PC genommen.

. L "name",xx

LOAD - Lade das Programm "NAME" von dem Peripheriegerät mit der Gerätenummer xx.

Gerätenummern sind:

01 Kassettenrekorder 1
02 Kassettenrekorder 2
08 Floppystation

Beispiel:

```
.L "BEISPIEL",01
```

..lädt das Programm "BEISPIEL" von Kassette in den Computerspeicher.

Beim Laden von Diskette kann vor dem Namen - von diesem durch Doppelpunkt getrennt - die Laufwerksnummer spezifiziert werden:

```
.L "1:BEISPIEL",08
```

..lädt das Programm "BEISPIEL" von der Diskette in Laufwerk 1

. M xxxx yyyy

MEMORY - Hexauflistung von Adresse xxxx bis yyyy.

Um den Inhalt einer Speicherzelle zu verändern, überschreiben Sie einfach das entsprechende Byte auf dem Bildschirm und übernehmen die Änderung durch Drücken der RETURN-Taste.

Beispiel:

```
M 0000 00FF
```

':' und 'Pfeil nach links' wirken genau wie bei normalem LIST.

. P xxxx

PRINTER DISASSEMBLER - Disassemblierung wie bei D auf einen Drucker.

Beispiel:

```
P A000
```

. R

REGISTER - Anzeige der Register des Mikroprozessors 6502.

Ausgegeben werden folgende Register:

```
PC = program counter   = Programmzähler
SR = status register   = Statusregister
AC = accumulator       = Akkumulator
XR = X register        = Indexregister X
YR = Y register        = Indexregister Y
SP = stack pointer     = Stackzeiger
```

Änderungen können Sie durch Überschreiben der Registerinhalte vornehmen.

. S "name",xx,yyyy,zzzz

SAVE - Abspeicherung von Maschinencode zwischen yyyy und zzzz auf das Peripheriegerät mit der Gerätenummer xx.

Beispiel:

```
.S "BEISPIEL",01,033A,034B
```

034B ist Endadresse plus eins, das letzte Datenbyte befindet sich also in 034A.

Beim Abspeichern auf Diskette sollten Sie immer eine Laufwerksnummer angeben (siehe .L)

```
.S "0:BEISPIEL",08,033A,034B
```

..speichert den Bereich zwischen 033A und 034A unter dem Namen "BEISPIEL" auf der Diskette in Laufwerk 0 ab.

. X

EXIT - Rückkehr von EXMON zu EXBASIC LEVEL II.

Um wieder in EXMON zu gelangen, steht Ihnen GO zur Verfügung.

SOFTMODULE

Sie haben die unglaubliche Fülle von Möglichkeiten gesehen, die Ihnen EXBASIC LEVEL II zur Verfügung stellt. EXBASIC LEVEL II gliedert sich in Hilfsfunktionen, Graphikbefehle, Mathematische Funktionen und LEVEL II Basic Befehle. Hinzu kommen je nach Serie die Sondermodule FAST TAPE, DOS SUPPORT, SCREEN SUPPORT und EXMON.

Nichtsdestotrotz mag es sein, daß Sie vielleicht die eine oder andere Funktion vermissen, von der Sie glauben, daß Sie sie schon sehr oft einsetzen würden, wenn Sie sie nur zur Verfügung hätten. Gerade ganz spezielle Anwendungsgebiete wie z.B. Matrixrechnung konnten wir bei der Entwicklung von EXBASIC LEVEL II beim besten Willen nicht berücksichtigen.

Deswegen haben wir uns etwas anderes ausgedacht: EXBASIC LEVEL II ist mit SOFTMODULEN erweiterbar! Das bedeutet für Sie, wenn Sie sich einmal EXBASIC LEVEL II zugelegt haben, so haben Sie Zugriff zu einer Auswahl an SOFTMODULEN, die Sie nach Belieben hinzufügen können. Damit ist es also möglich, daß Sie EXBASIC LEVEL II auf Ihrem ganz speziellen Anwendungsgebiet "stark machen".

SOFTMODULE werden nicht in Eproms in den Computer eingesteckt, sondern sind reine Software. Sie stehen im oberen Bereich des RAM und belegen also Computer-speicherkapazität. Da Sie ein SOFTMODUL aber nur benutzen werden, wenn Sie es wirklich benötigen, rechtfertigen die leistungsfähigen SOFTMODUL-Funktionen, die Ihnen ein SOFTMODUL zur Verfügung stellt, den Einsatz eines entsprechenden SOFTMODULS. Oftmals erspart Ihnen eine einzige Funktion eines SOFTMODULS ein komplexes Unterprogramm und somit letztendlich Speicherkapazität, ganz abgesehen von der Zeit, die Sie zur Entwicklung des speziellen Unterprogrammes benötigten.

Die einzelnen SOFTMODULE sind selbstverständlich miteinander kombinierbar. Sie können also z.B. ein Graphikmodul und eines für mehrfachgenaue Arithmetik gleichzeitig benutzen. Mit anderen Worten: Sie allein entscheiden, wie weit Sie EXBASIC LEVEL II ausbauen wollen, ganz nach Ihren Wünschen.

Zu EXBASIC LEVEL II werden zwei kleine SOFTMODULE mitgeliefert: CLR/SORT ARRAY und GOTO/GOSUB EXPR. Sie finden beide SOFTMODULE anschließend abgedruckt. In Zukunft werden EXBASIC LEVEL II SOFTMODULE von uns direkt oder über den örtlichen Computerfachhandel und Buchhandel angeboten werden. Ihr Fachgeschäft wird Ihnen gerne die jeweiligen Neuheiten vorstellen.

Allen SOFTMODUL-Funktionen und -befehlen ist gemeinsam, daß sie mit dem EXBASIC LEVEL II Kommando CALL aufgerufen werden. Dadurch wird die Ausführungszeit des Basicinterpreters durch SOFTMODULE nicht verlangsamt, unabhängig davon, wieviele SOFTMODULE gerade aktiv sind.

Alle SOFTMODUL Befehle weisen die gleiche Syntax auf, sie sieht wie folgt aus:

```
CALL ( befehlswort , argumentliste)
```

Um z.B. den Befehl SORT aufzurufen, geben Sie...

```
CALL (SORT, A)
```

..ein. SORT ist das Befehlswort, A ein Argument. Mehrere Argumente werden durch Komma voneinander getrennt.

Die SOFTMODULE bringen im allgemeinen eigene Fehlermeldungen mit. EXBASIC LEVEL II kennt aber auch die Meldung ?MODUL ERROR.

Um zwei oder mehrere SOFTMODULE zu koppeln, gehen Sie wie folgt vor: Laden Sie das erste Modul und starten Sie es mit RUN. Laden Sie danach das zweite Modul und starten Sie auch dieses wieder mit RUN. Fertig - jetzt steht Ihnen der Befehlsvorrat beider SOFTMODULE gleichzeitig zur Verfügung. So einfach haben wir das gemacht.

Beachten Sie bei der Benutzung von SOFTMODULEN aber unbedingt, daß Sie nachdem Sie ein oder mehrere MODUL(E) mit RUN aktiviert haben, nicht mehr das Kommando DEF CALL verwenden dürfen. CALL steht danach also nur noch zum Aufrufen der SOFTMODUL Funktionen zur Verfügung.

CLR/SORT ARRAY SOFTMODUL

Dieses Modul stellt Ihnen die zwei Befehle SORT und CLR zur Verfügung.

* CALL(SORT , feldvariablenname)

Hin und wieder kommt man beim Programmieren einfach nicht darum herum, ein Variablenfeld zu sortieren. Das mögen Zahlen, Namen oder sonst etwas sein. Wenn Sie schon einmal einen Sortieralgorithmus geschrieben haben, dann wissen Sie, wie langsam diese sind.

Hier hilft Ihnen der Befehl SORT. SORT sortiert ein endimensionales Variablenfeld in Sekundenschnelle.

Beispiele:

```
CALL (SORT, A)      sortiert A(0) bis A(..)
CALL (SORT, A%)     sortiert A%(0) bis A%(..)
CALL (SORT, A$)     sortiert A$(0) bis A$(..)
```

'SORT, A' hat also nichts mit der Variablen A zu tun, sondern bezieht sich immer auf das Feld A(0) bis A(..).

Die Sortierzeit ist abhängig von der Feldgröße und von der Art des Feldes, bei Stringfeldern auch von der Länge der einzelnen Strings. Am schnellsten sortiert wird ein Integerfeld (%), am langsamsten ein Stringfeld ($).

Beispiel:

'SORT,X' mit X(0) bis X(100) braucht ca. 1.13 Sekunden
'SORT,X%' mit X%(0) bis X%(100) braucht ca. 0.43 Sekunden

Wenn Sie probieren, ein Feld zu sortieren, das nicht existiert, so erfolgt die Fehlermeldung ?ARRAY ERROR.

* CALL(CLR , feldvariablenname)

Sie wissen, daß es im normalen Basic nicht möglich ist, ein einmal festgelegtes Feld wieder zu löschen. Es sei denn, Sie geben CLR ein, aber dann sind alle Variablen gelöscht, also auch die, die noch gebraucht werden. Durch diesen Umstand liegt oft sehr viel Speicherplatz brach. "Das muß nicht sein", haben wir uns gedacht und den Befehl SOFTMODUDL CLR kreiert.

SOFTMODUL CLR löscht ein einzelnes Variablenfeld, ohne dabei andere Variablen mit zu zerstören. Die Dimension des Feldes spielt keine Rolle.

Beispiele:

A(0) bis A(100)   wird mit CALL(CLR,A) gelöscht
X%(0,0) bis X%(10,5)   wird mit CALL(CLR,X%) gelöscht
S$(0,0,0) bis S$(12,2,10) wird mit CALL(CLR,S$) gelöscht

Machen Sie sich klar, daß die SOFTMODUL-Funktion CLR nichts mit dem normalen CLR-Befehl zu tun hat. Falls das mit SOFTMODUL CLR angegebene Feld nicht existiert, erfolgt der Ihnen schon von SORT bekannte ?ARRAY ERROR.

Beachten Sie: Wenn Sie mit SOFTMODUL CLR ein Stringfeld löschen, so wird zunächst nur der Speicherplatz für die Feldindizes, nicht aber für die Strings freigegeben. Sie sehen dies bei einer Auflistung des Computerspeichers mit MEM. Unter ARRAYS finden Sie dann den Wert 0 (falls kein anderes Feld mehr vorhanden ist), während bei STRINGS noch die volle Bytezahl der Strings des Feldes angegeben ist. Um auch den Platz für die Strings dem Basicspeicher wieder zuzuschlagen, benutzen Sie FRE(0). FRE(0) ruft im Unterschied zu MEM die sog. Garbage-Collect-Routine des Computers auf und diese Routine wiederum "befreit" den Speicher von unnötigem Ballast, also auch von den Strings, die nach der Feldlöschung nicht mehr gebraucht werden. Wenn Sie sich nach FRE(0) den Computerspeicher mit MEM erneut ansehen, so werden Sie feststellen, daß dann auch der entsprechende Stringspeicherplatz freigegeben worden ist.

Beispiel zu SOFTMODUL CLR mit Stringfeld:

```
100 CALL (CLR, X$) : A=FRE(0)
```

..löscht das Feld X$ vollständig und gibt auch die Strings frei. Bei den Serien 2001 und 3001 kann FRE(0) u.U. mehrere Minuten dauern, bedingt durch das Betriebssystem dieser Computer.

GOTO/GOSUB EXPR SOFTMODUL

Mit diesem SOFTMODUL können Sie berechnete GOTO- und GOSUB-Sprünge programmieren.

* CALL(GOTO , numerischer ausdruck)

Dieser Befehl bewirkt einen Sprung zu der Zeilennummer, die durch den Wert des numerischen Ausdrucks festgelegt ist. Die Zeilennummer darf zwischen 0 und 63999 liegen. Als Zeilennummer nimmt der Befehl nur den ganzzahligen Teil des numerischen Ausdruck, es ist also quasi INT impliziert.

Alle Fehlermeldungen, die im Zusammenhang mit SOFTMODUL GOTO auftreten können, entsprechen denen des normalen GOTO Befehls.

Beispiel zu SOFTMODUL GOTO:

```
100 INPUT A
110 CALL (GOTO, A)
```

Es wird zu der Zeilennummer gesprungen, die bei INPUT eingegeben wird.

* CALL(GOSUB , numerischer ausdruck)

Mit dem GOTO/GOSUB EXPR SOFTMODUL ist es auch möglich, berechnete Unterprogrammsprünge zu programmieren. Alles über den SOFTMODUL Befehl GOTO Gesagte gilt entsprechend auch für SOFTMODUL GOSUB.

Beispiel:

```
100 CALL (GOSUB, X*50+1000)
```

Dabei wird angenommen, daß der Variablen X vorher ein bestimmter Wert zugewiesen wurde. Falls X gleich 0 ist, erfolgt ein Unterprogrammsprung zu Zeile 1000.

Hier nun die Listings für die beiden SOFTMODULE 'CLR/SORT ARRAY' und 'GOTO/GOSUB EXPR'.

```
10 ' CLR / SORT ARRAY - SOFTMODUL - (C) 1981 BY M.KRAUSE
20 L=522:HIMEMDEEK(52)-L:L=521:S=DEEK(52):I=S:SE=PEEK(36886)-48
30 FORI=STOS+L:READA$:J=DEC(A$):POKEI,J:Z=Z+J:NEXT:D=215
40 IFZ<>66213PRINT"N?CHECKSUM ERROR]":END
50 IFDEEK(1021)<32768DOKES+D,DEEK(1021)ELSEDOKES+D,48896:IFSE<4DOKES+D,52739
60 DEFCALL=S+130:DOKES+135,S+217:DOKES+177,S:DOKES+280,S+217
70 DOKES+348,S+422:DOKES+398,S+68:IFSE>3END
80 DOKES+215,52739:DOKES+221,52728:DOKES+224,53101:DOKES+274,50007
90 DOKES+277,53536:DOKES+459,55982:DOKES+466,56167
100 POKES+176,44:POKES+397,44:HIMEMDEEK(52)+130
110 DATAA0,04,B1,5C,18,0A,69,05,65,5C,85,B4,A5,5D,69,00,85,B5,A0,02,B1,B4,C5
120 DATA31,90,14,85,B7,88,B1,B4,85,B6,88,B1,B4,F0,08,A8,91,B6,C8,A9,FF,91,B6
130 DATAA5,B4,18,69,03,85,B4,90,02,E6,B5,C5,57,D0,D5,A5,B5,C5,58,D0,CF,60,A4
140 DATA6E,C0,03,D0,F9,88,B1,40,C5,31,90,16,85,B5,88,B1,40,85,B4,88,B1,40,A8
150 DATAA5,40,91,B4,C8,A5,41,91,B4,A0,02,B1,5C,C5,31,90,14,85,B5,88,B1,5C,85
160 DATAB4,88,B1,5C,A8,A5,5C,91,B4,C8,A5,5D,91,B4,60,C9,9C,D0,3E,20,67,6B,A5
170 DATA2F,85,58,C8,A5,2E,38,F1,5C,85,2E,C8,A5,2F,F1,5C,85,2F,38,E5,58,AA,E8
180 DATA88,B1,5C,18,65,5C,85,57,C8,B1,5C,65,5D,85,58,20,8E,6A,A0,00,B1,57,91
190 DATA5C,C8,D0,F9,E6,5D,E6,58,CA,10,F2,60,C9,53,D0,0E,20,70,00,C9,B0,D0,07
200 DATA20,70,00,C9,54,F0,41,4C,00,BF,20,70,00,20,F5,BE,20,2B,C1,A6,2C,A5,2D
210 DATA86,5C,85,5D,C5,2F,D0,04,E4,2E,F0,1D,A0,00,B1,5C,C8,C5,42,D0,06,A5,43
220 DATAD1,5C,F0,C2,C8,B1,5C,18,65,5C,AA,C8,B1,5C,65,5D,90,D7,A2,80,4C,CF,B3
230 DATA4C,70,C3,20,51,6A,A0,04,B1,5C,C9,01,D0,F2,A0,00,A2,05,B1,5C,10,01,CA
240 DATAC8,B1,5C,10,02,CA,CA,86,6E,A0,02,B1,5C,18,65,5C,85,46,C8,B1,5C,65,5D
250 DATA85,47,A5,5C,18,69,07,85,5C,90,02,E6,5D,A5,5C,A6,5D,85,40,86,41,85,80
260 DATA86,81,20,1E,6B,10,08,A5,80,A6,81,85,40,86,41,A5,80,18,65,6E,85,80,90
270 DATA02,E6,81,C5,46,90,E4,A5,81,C5,47,90,DE,A4,6E,88,B1,40,AA,B1,5C,91,40
280 DATA8A,91,5C,88,10,F3,20,BC,69,A5,5C,18,65,6E,85,5C,90,02,E6,5D,C5,46,90
290 DATAB0,A5,5D,C5,47,90,AA,60,A5,6E,C9,05,F0,1A,C9,03,F0,24,A0,00,B1,40,D1
300 DATA80,D0,07,C8,B1,40,F1,80,F0,06,A9,01,90,02,A9,FF,60,A5,80,A4,81,20,D8
310 DATACC,A5,40,A4,41,4C,91,CD,A0,02,B1,80,99,5E,00,88,10,F8,A0,02,B1,40,99
320 DATA69,00,88,10,F8,AA,38,E5,5E,F0,08,A9,01,90,04,A6,5E,A9,FF,85,63,A0,FF
330 DATAE8,C8,CA,F0,08,B1,6A,D1,5F,F0,F6,D0,B8,A5,63,60
```

```
10 ' GOTO / GOSUB EXPR - SOFTMODUL - (C) 1981 BY M.KRAUSE
20 L=58:HIMEMDEEK(52)-L:L=57:S=DEEK(52):I=S:SE=PEEK(36886)-48
30 FORI=STOS+L:READA$:POKEI,DEC(A$):NEXT:D=S+9
40 IFDEEK(1021)<2↑15DOKED,DEEK(1021)ELSEIFSE>3DOKED,48896ELSEDOKED,52739
50 DEFCALL=S:IFSE>3END
60 DOKES+19,52728:DOKES+22,52363:DOKES+25,54994
70 DOKES+28,51120:DOKES+36,49947
80 DATAC9,89,F0,07,C9,8D,F0,19,4C,29,6D,68,68,68,68,20,70,00,20,F5,BE,20,84
90 DATABD,20,2D,C9,20,33,B8,4C,10,91,A9,03,20,93,B3,68,68,A5,78,48,A5,77,48
100 DATAA5,37,48,A5,36,48,A9,8D,48,B8,50,D5
```



Beispielprogramme

Die abgedruckten Beispielprogramme dienen dazu, Ihnen die vielfältigen Möglichkeiten von EXBASIC LEVEL II zu demonstrieren und geben Ihnen gleichzeitig eine Hilfe beim Entwerfen eigener Programme.

```
10 'CRAZY LINES
20 LETTEROFF:PRINT"]":SE=PEEK(36886)-48:BS=79-80*(SE=8)
30 V1=V2:H1=H2:GETA$:IFA$>""IFA$="@"HARDCOPYELSEPRINT"]"
40 V2=RND(49):H2=RND(BS):A=H2-H1:B=V2-V1:IFA=0ANDB=0THEN.
50 IFABS(A)>ABS(B)C=B/A:FORI=H1TOH2STEPSGN(A):SET(I,V1+(I-H1)*C):NEXT:GOTO30
60 C=A/B:FORI=V1TOV2STEPSGN(B):SET(H1+(I-V1)*C,I):NEXT:GOTO30
```

```
10 'CARPET
20 PRINT"]":LETTEROFF:SE=PEEK(36886)-48:WD=39-40*(SE=8)
30 A=RND(WD/5+1)*5-5:B=RND(6)*4-4
40 SPACEA,B,A+4,B+3,RND(63)+64+128*(RND(2)-1):GOTO30
```

```
10 'SPIRALE
20 PRINT"]":S8=PEEK(36886)-48=8:B=80*(1-S8)-1:D=49:P=2-2*S8
30 FORI=ATOB:SET(I,C):NEXT:A=A+P:FORI=CTOD:SET(B,I):NEXT:FORI=BTOASTEP-1
40 SET(I,D):NEXT:C=C+2:FORI=DTOCSTEP-1:SET(A,I):NEXT:B=B-P:D=D-2:IFC<DTHEN30
```

```
10 '6 BIT ASCII TEST
20 LETTEROFF:BS=39:REM 79 FUER 8001
30 FORI=0TO255:SPACE0,0,BS,24,I:PRINT@0I"█ ":SEC1:NEXT
```

```
10 'GAUSSSCHE GLOCKENKURVE
20 LETTEROFF:SE=PEEK(36886)-48:WD=40-40*(SE=8)
25 DIMA(WD):BS=(WD-1)/10:E=201:Q=24*WD:PRINT"□"
30 S=0:FORI=0TOBS:S=S+RND(9):NEXT:Z=A(S)
40 IFZ<EPRINT@Q+S;:VPLOT(Z):A(S)=Z+1:GOTO30ELSEPRINT@0
```

```
10 'HISTOGRAMM
20 LETTEROFF:SE=PEEK(36886)-48:BS=40*(1-(SE=8))
30 PRINT"□HISTOGRAMM▩▩▩▩":DIMD(BS)
40 INPUTFORM"▩▩  F(X)=";A$:IFA$=""ENDELSEPRINT"□"
50 INPUTFORM"▩ANFANG=";B$,10:PRINT@BS/2;:INPUTFORM"ENDE=";C$,10
60 A=EVAL(B$):E=EVAL(C$):K=0:L=1E38:H=-L:IFA=ETHEN40
70 FORX=ATOESTEP(E-A)/BS:J=EVAL(A$):D(K)=J:K=K+1:H=MAX(H,J):L=MIN(L,J):NEXT
80 PRINT"□":PRINT@24*BS;:D=192/(H-L):FORI=0TOBS-1:VPLOT(D(I)-L)*D:NEXT:GOTO50
```

```
10 'MINI CURSOR
20 PRINT"□";:BS=79:'159 FUER 8001
30 IFPOINT(S,Z)RESETS,ZELSESETS,Z
40 GETA$:IFA$=""THEN.
50 IFA$="8"IFZ>0Z=Z-1
60 IFA$="2"IFZ<49Z=Z+1
70 IFA$="6"IFS<BSTHENS=S+1ELSEIFZ<49S=0:Z=Z+1
80 IFA$="4"IFS>0S=S-1ELSEIFZ>0S=BS:Z=Z-1
90 IFA$="▩"S=0:Z=0
100 IFA$="□"RUNELSE30
```

```
10 'PRIMFAKTOREN
20 INPUT"▩ZAHL";Z:IFFRAC(Z)>0ORZ<=0THEN.ELSEPRINT:PRINTZ"=";
30 IFODD(Z)T=3:W=SQR(Z):GOTO50ELSEPRINT" 2 *";:Z=Z/2:GOTO.
40 PRINTT"*";:Z=Z/T:W=SQR(Z)
50 IFFRAC(Z/T)=0THEN40ELSEIFZ>1T=T+2:IFT>WPRINTZELSE.ELSEPRINT"▮▮ "
```

EXBASIC LEVEL II™

Hier finden Sie ein vollständiges Spielprogramm in EXBASIC LEVEL II abgefaßt. Zur Benutzung folgende Hinweise:

Businesstastatur: Start mit 'J', Bewegung mit 'N' und 'M'
Graphiktastatur: Start mit 'HOME', Bewegung mit '.' und '='

```
10 ' ++ BREAK OUT 2.1 ++
20 ' (C) 4/80 M.KRAUSE
30 FORA=0TO3:READR(A):NEXT:C$=STRING$(40,"-"):S=826
40 READA$:IFA$<>"@"POKES,DEC(A$):S=S+1:GOTO.
50 POKE32768,96:IFPEEK(33792)=96DOKE878,58926ELSESCREEN*
60 F$="######":Y=151:S=32:L=32:U=120:T=33167:V=81:X=50:Z=74:Q=40
70 M=17100:D=1:P=0:O=32967:E=207
80 D=D-1:H=40:C=33668
90 PRINTUSING"▫ ▩BALL■ # ▩HIGHSCORE■ "+F$+" ▩POINTS■: "+F$,D,A(1),P
100 PRINT" ╭"STRING$(38,"-")"╮";
110 A$="|▩┳┳┳┳┳┳┳┳┳┳┳┳┳┳┳┳┳┳■|":PRINTA$A$A$;
120 B$="|"+STRING$(38,32)+"|":PRINTB$B$A$A$A$B$B$A$A$A$;
130 FORI=1TO7:PRINTB$;:NEXT:PRINT"|"STRING$(18,32)"▔▔"STRING$(17,32)"|";
140 PRINT"╰──────────╮"STRING$(16,".")"╭──────────╯▮▮▮-";:POKE189,17
150 IFNOTGPRINT@726,USING"▩GAME OVER        SCORE="+F$,SC
160 D=D+1:PRINT@5,D:A=33608+RND(38):R=-39:F=1
170 A=A+R:B=PEEK(A):IFB=SPOKEA,V:GOTO280ELSEIFB<EGOTO210
180 F=3-F:POKEA,S:IFB=EPOKEA+1,S:A=A+1ELSEPOKEA-1,S
190 H=H-.1:P=P+X:IFA<TTHENP=P+X:IFA<OTHENP=P+X
200 PRINT@33,USINGF$,P:IFP<MGOTO230ELSEM=M+17100:GOTO80
210 IFB<>UGOTO240
220 A=A-Q:PL=PEEK(Y)AND3:IFPL=2F=0ELSEIFPL=1F=1ELSEF=3-F
230 R=R(F):GOTO170
240 IFB=46POKE158,0:IFD=5GOTO320ELSEWAIT158,1:GETA$:GOTO150
250 IFB=64F=3-F:GOTO230ELSEIFB=93F=5-FAND3:GOTO230
260 IFB<128F=F+2AND3:GOTO230ELSEIFF<2GOTO230ELSEPOKEC-1,S
270 POKEC,S:POKEC+1,S:C=A+4*R:POKEC-1,U:POKEC,U:POKEC+1,U:GOTO170
280 IFGFORI=0TOH:NEXT:GOTO300
290 IFPEEK(Y)=ZTHENG=-1:PRINT@726,STRING$(28,32):DOKE144,826:GOTO70
300 IFPEEK(A)<>UTHENPOKEA,S
310 GOTO170:DATA-41,-39,41,39
320 SC=P:SEC1:IFP<A(5)GOTO380
330 R=1:FORI=5TO1STEP-1:IFA(I)>PTHENR=I+1:I=0
340 NEXT:OFF:PRINT"▫"C$:PRINT"    ▩GAME OVER■   SCORE:"P:PRINT:PRINTC$
350 PRINTUSING"▩YOU ARE ON THE #. PLACE!",R
360 FORI=5TORSTEP-1:A$(I+1)=A$(I):A(I+1)=A(I):NEXT:A(R)=P
370 INPUTFORM"▩ENTER YOUR NAME: ";A$(R),15
380 PRINT"▫"C$"▩THE FIVE ▩BESTS■ ARE:▩▩
390 FORI=1TO5:PRINTA$(I);STRING$(20-LEN(A$(I)),32);:PRINTUSING"######",A(I)
400 NEXT:OFF:PRINTC$:G=0:FORI=1TO1E3:GETA$:IFA$="▩"DISPOSENEXTELSENEXT
410 GOTO60:DATAA5,8F,29,02,D0,2D,A6,BD,A5,97,30,27,4A,90,11,E0,22,B0,20
420 DATAE8,A9,20,9D,71,83,A9,78,9D,74,83,D0,11,4A,90,10,8A,F0,0D,CA,A9,20
430 DATA9D,75,83,A9,78,9D,72,83,86,BD,4C,55,E4,@
```

Befehlsliste

Nachfolgend finden Sie eine Liste aller Befehle, die Ihnen nach dem Aktivieren von EXBASIC LEVEL II zur Verfügung stehen, sofern sie einen Interpretercode besitzen. Dies ist nicht der Fall bei den DOS SUPPORT Kurzkommandos und den EXMON Befehlen. Deshalb finden Sie diese auch nicht aufgeführt.
Diese Liste sagt auch nichts aus über mögliche Kombinationen der einzelnen Befehlsworte. So steht z.B. nur DEF in der Liste, dagegen sind die Befehlskombinationen DEF FN, DEF USR, DEF CALL nicht aufgeführt. Vielmehr stehen FN, USR und CALL extra.

Die Befehlsliste ist wie folgt aufgebaut: Links steht das Befehlswort ausgeschrieben, in der Mitte die mögliche Abkürzung, die Sie beim Eintippen verwenden dürfen und rechts schließlich der Interpretercode.

Jedes Basic und EXBASIC LEVEL II Befehlswort wird intern komprimiert durch eine einzige Zahl zwischen 1 und 255 abgespeichert. Diese Zahl nennen wir den Interpretercode des betreffenden Befehls. Der Vorteil liegt in der großen Ersparnis an Speicherkapazität. So wird z.B. statt der 9 Buchstaben INPUTFORM nur der Code 234 im Computerspeicher abgelegt. Wenn der Computer beim Listen auf die Zahl 234 stößt, gibt er INPUTFORM auf dem Bildschirm oder auch Drucker aus. Sie brauchen sich normalerweise um den Interpretercode gar nicht zu kümmern. Er ist aber der Vollständigkeit zuliebe aufgeführt.

Die Abkürzungen können Sie beim Eintippen eines Programmes oder auch im Direktmodus benutzen. Wenn Sie eine Programmzeile mit Abkürzungen eingeben, so werden diese bei LIST oder ähnlichen Befehlen (FIND etc.) in jedem Fall ausgeschrieben. Für das Programm selbst spielt es also überhaupt keine Rolle, ob es mit Abkürzungen eingetippt wurde oder nicht. Insbesondere können Sie auf diese Weise auch keinen Speicherplatz sparen. Es ist lediglich eine Erleichterung beim Eintippen.

Sie sehen dies am einfachsten, wenn Sie einmal diese Programmzeile...

```
100 ?:?:?:?:?:?:?:?:?
```

..eingeben. '?' ist die Abkürzung für PRINT. Geben Sie nun LIST ein und auf dem Bildschirm erscheint...

```
100 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
```

Beachten Sie bei den Abkürzungen bitte unbedingt: Alle Großbuchstaben müssen ohne SHIFT eingegeben werden, unabhängig davon, ob sie dann auf dem Bildschirm groß oder klein erscheinen. Der jeweils letzte Buchstabe der Abkürzung wird mit SHIFT eingegeben. Auch dabei spielt die Darstellung auf dem Bildschirm keine Rolle.

Die Zeichen vor den Befehlen, sofern vorhanden, bedeuten:

- 9 nur für die Serien 2001/3001/4001 (2+3+4 = 9)
- 12 nur für die Serien 4001/8001 (4+8 = 12)
- 8 nur für die Serie 8001
- A Abkürzung ist nicht möglich

Sie finden bei allen Befehlen rechts noch genug Platz, um dort eigene Notizen unterzubringen. Sie können sich dort z.B. mögliche Befehlskombinationen oder -bedeutungen notieren. Nutzen Sie den Freiraum.

Befehlsliste

	ABS	Ab	182
	AND	An	175
12	APPEND	Ap	212
	ASC	As	198
	ATN	At	193
	AUTO	Au	5
12	BACKUP	BAc	210
	BASIC	Ba	223
8	BEGINLINE	Be	19
	BEEP	BEe	227
	CALL	Ca	8
12	CATALOG	CAt	215
	CHR$	Ch	199
	CLOSE	CLo	160
	CLR	Cl	156
	CMD	Cm	157
12	COLLECT	COl	209
12	CONCAT	CONc	204
	CONT	Co	154
12	COPY	COp	211
A	COS	COS	190
	DATA	Da	131
12	DCLOSE	Dc	206
A	DEC	DEC	252
	DEEK	DEe	242
A	DEF	DEF	150
	DEL	De	4
8	DELLINE	DEl	18

A	DIM	DIM	134
12	DIRECTORY	DIr	218
	DISPOSE	Di	230
12	DLOAD	Dl	214
	DOKE	Do	13
12	DOPEN	DOp	205
8	DOWN	DOw	22
12	DSAVE	Ds	213
	DUMP	Du	6
	ELSE	El	239
A	END	END	128
8	ENDLINE	En	20
	ERROR	Er	240
	EVAL	Ev	254
	EXEC	Ex	10
A	EXP	EXP	189
	FAST	Fa	228
	FIND	Fi	3
A	FN	FN	165
	FOR	Fo	129
	FRAC	Fr	250
A	FRE	FRE	184
	GET	Ge	161
A	GO	GO	203
	GOSUB	GOs	141
	GOTO	Go	137
	HARDCOPY	Ha	233
12	HEADER	HEa	208
	HELP	He	226
	HEX$	HEx	253
	HIMEM	Hi	232
	HPLOT	Hp	12
A	IF	IF	139
A	INPUT	INPUT	133
	INPUTFORM	INPUTf	234
	INPUTLINE	INPUTl	15
A	INPUT#	INPUT#	132
8	INSTLINE	In	17
	INSTR	INs	245
A	INT	INT	181
A	LEN	LEN	195
A	LET	LET	136
	LETTER	Le	225
	LIST	Li	155
	LOAD	Lo	147
A	LOG	LOG	188

	MATRIX	Ma	229
A	MAX	MAX	248
A	MEM	MEM	221
	MERGE	Me	11
	MID$	MId	202
	MIN	Mi	247
9	MOD	Mo	235
A	NEW	NEW	162
	NEXT	Ne	130
	NOT	No	168
	ODD	Od	251
	OFF	Of	1
A	ON	ON	145
	OPEN	Op	159
A	OR	OR	176
	PEEK	Pe	194
	POINT	Po	244
	POKE	POk	151
A	POS	POS	185
	PRINT	?	153
	PRINT@	Pr	231
A	PRINT#	PRINT#	152
	READ	REa	135
12	RECORD	REc	207
A	REK	REK	9
A	REM	REM	143
12	RENAME	RENa	216
	RENUM	Re	2
	RESET	REs	220
	RESTORE	RESt	140
	RESUME	RESu	224
	RETURN	REt	142
	RIGHT$	Ri	201
	RND	Rn	246
	ROUND	Ro	241
	RUN	Ru	138
	SAVE	Sa	148
12	SCRATCH	SCRa	217
8	SCREEN	Sc	235
A	SEC	SEC	238
	SET	Se	16
	SGN	Sg	180
	SIN	Si	191
	SPACE	Sp	14
	SPC(	SPc	166
	SQR	Sq	186
	STEP	STe	169
	STOP	STo	144

A	STR$	STR$	196
	STRING$	St	243
	SWAP	Sw	236
	SYS	Sy	158
	TAB(	Ta	163
A	TAN	TAN	192
	THEN	Th	167
A	TO	TO	164
	TRACE	Tr	222
8A	UP	UP	21
	USING	USING	237
A	USR	USR	183
A	VAL	VAL	197
	VARPTR	Va	249
	VERIFY	Ve	149
	VPLOT	Vp	7
	WAIT	Wa	146
	+		170
	-		171
	*		172
	/		173
	↑		174
	)		177
	=		178
	(		179

Wie Sie an dieser Liste erkennen können, ist EXBASIC LEVEL II bis auf MOD voll aufwärtskompatibel, jedoch nur mit Einschränkung abwärtskompatibel. Mit anderen Worten: Ein in EXBASIC LEVEL II geschriebenes Programm für eine untere Serie ist immer auch auf Computern höherer Serien ablauffähig. In umgekehrter Richtung ist die Kompatibilität dagegen nur dann gegeben, wenn das Programm keinen mit '12' bzw. '8' gekennzeichneten Befehl enthält. Ein für 4001 geschriebenes Programm läuft also nur dann auf 2001/3001, wenn es keine 12er Befehle enthält, ein für 8001 gedachtes Programm nur dann auf 4001, wenn es keine 8er Befehle beinhaltet und nur dann auf 2001/3001, wenn es weder 8er noch 12er Befehle hat. Zwischen den Serien 2001 (mit neuen Roms) und 3001 bestehen keine Unterschiede. Diese verschiedenen Kompatibilitätsstufen in EXBASIC LEVEL II resultieren aus der Berücksichtigung der unterschiedlichen Möglichkeiten und Erfordernisse der einzelnen CBM-Serien.

Notizen

Nutzen Sie diese und die folgenden Seiten, um Erfahrungen, die Sie mit EXBASIC LEVEL II machen und Informationen, die Sie in Veröffentlichungen finden, niederzuschreiben. Dadurch bleibt dieses Handbuch immer auf dem neuesten Stand.

Um Sie zur Benutzung der Notizseiten anzuregen, haben wir selbst erste Notizen eingetragen. Fügen Sie weitere hinzu!

* Warmstart mit SYS

EXBASIC LEVEL II muß vor der ersten Benutzung mit SYS gestartet werden. Nun wird beim Aktivieren von EXBASIC LEVEL II mit SYS 37100 das im Speicher stehende Programm gelöscht und die EXBASIC LEVEL II Meldung auf dem Bildschirm ausgegeben. Für kommerzielle Anwendungen ist das ungeeignet.

Abhilfe: Warmstart mit SYS 37139. Mit diesem Befehl wird EXBASIC LEVEL II aktiviert, ohne daß dadurch das im Speicher stehende Programm gelöscht wird. Es erfolgt auch keine Meldung auf dem Bildschirm. Die Programmausführung wird normal fortgesetzt.
SYS 37139 muß der erste Befehl des Programmes sein, das mit RUN eingeladen wird.
Aber Achtung: Beim Eingeben des Programmes muß EXBASIC LEVEL II unbedingt aktiv sein. Andernfalls kommt es bei einem späteren Ablauf zu ?SYNTAX ERROR, sobald ein EXBASIC LEVEL II Befehl auftritt. Daran ändert auch der beim Ablauf vorkommende Warmstart nichts. Also: Vor dem Programmieren schon EXBASIC LEVEL II einschalten.

* Compiler

Es ist möglich, in EXBASIC LEVEL II geschriebene Programme zu compilieren, und zwar mit dem DTL-Compiler von Drive Technology Ltd. Dabei ist jedoch folgendes zu beachten:

Der DTL-Compiler verkraftet keine Statements, die mit einem Wort des normalen Commodore Basics beginnen und gleichzeitig ein EXBASIC LEVEL II Wort enthalten. So arbeiten z.B. diese Befehle...

INPUTFORM, INPUTLINE, BEEP, SWAP, SPACE, HELP

..und die meisten anderen einwandfrei nach der Compilierung, da sie eigenständige Worte von EXBASIC LEVEL II sind.

Hingegen kann der DTL-Compiler z.B. ...

PRINT USING, ON ERROR GOTO, DEF CALL

..und andere Befehle nicht direkt verarbeiten. In jedem der drei Fälle erkennt der Compiler das erste Befehlswort als normales Commodore Basicwort (PRINT, ON, DEF) und ist nicht in der Lage, die folgenden EXBASIC LEVEL II Befehlsworte (USING, ERROR, CALL) zu verarbeiten.

Abhilfe: Setzen Sie solche Kommandos, die der Compiler nicht direkt verarbeiten kann, in den EXEC-Befehl ein. Dadurch wird gewährleistet, daß das betreffende Statement mit einem EXBASIC LEVEL II Befehl beginnt und die Compilierung läuft ohne Schwierigkeiten.

Beispiel:

Diese Zeile...

```
100 PRINT USING F$,X
```

..kann der Compiler nicht verarbeiten. Ersetzen Sie die Zeile durch...

```
100 EXEC ("PRINT USING F$,X")
```

..und die Compilierung bereitet keine Probleme mehr.

Mit FIND können Sie alle kritischen EXBASIC LEVEL II Befehle eines Programmes heraussuchen oder mit dem SOFTMODUL REPLACE automatisch mit EXEC versehen.

Beachten Sie, daß auch compilierte Programme - genau wie normale - nur bei aktivem EXBASIC LEVEL II lauffähig sind. Während des Compilierungvorganges dagegen darf EXBASIC LEVEL II nicht eingeschaltet sein.





NOTIZEN 7



Stichwortverzeichnis

Abkürzungen der BASIC-Befehle 99
Ausschalten von EXBASIC LEVEL II 24
AUTO 14

BASIC 24
BEEP (OFF) 63
BEGINLINE 82

CALL 52
CLR (SOFTMODUL) 90

DEC 36
DEEK 56
DEF CALL 51
DEF USR 51
DEL 15
DELLINE 80
DISPOSE 45
DOKE 55
DOS SUPPORT 77
DOWN 82
DUMP 17

Einschalten von EXBASIC LEVEL II 9
ELSE 37
ELSE, Restriktionen 71
ENDLINE 81
ERROR 64
EVAL 61
EXEC 62
EXMON 85

FAST (OFF) 18
FAST TAPE 73
FILE NOT FOUND ERROR 69
FIND 13
FORMAT ERROR 69
FRAC 34

GO (EXMON) 85
GO (TIM) 20
GOSUB EXPR (SOFTMODUL) 91
GOTO EXPR (SOFTMODUL) 91

HARDCOPY 68
HELP 23
HELP* 24
HEX$ 36
HIMEM 20
HPLOT 28

IF..THEN..ELSE 37
INPUTFORM 49
INPUTLINE 48
INSTLINE 80
INSTR 60
Interpretercode 99

LETTER (OFF) 18
LIST 22
LOAD 73
LOAD* 73
LOAD ERROR 69

MATRIX 17
MAX 33
MEM 19
MERGE 25
MERGE* 25
MIN 33
MOD 76
MODUL ERROR 69
MUSIKBOX 63

NEXT, Restriktionen 71

ODD 35
OFF 16
ON 16
ON ERROR GOTO 64
ON..RESTORE 41

POINT 31
PRINT@ 27
PRINT USING 42
PUNKT '.' 18
PUNKT '.', Restriktionen 71

REK 44
REM 68
RENUM 15
RESET 30
RESTORE 40
RESUME 66
RESUME WITHOUT ERROR 69
RND 35
ROUND 34

SAVE 75
SAVE* 75
SCREEN 79
SCREEN* 83
SCREEN DOWN 82
SCREEN SUPPORT 79
SCREEN UP 82
SEC 63

SET 30
SOFTMODULE 89
SORT (SOFTMODUL) 90
SPACE 57
STOP ON/OFF 23
STRING$ 59
SWAP 68

THEN 37
TRACE (OFF) 16

UP 82
USING 42
USR 51

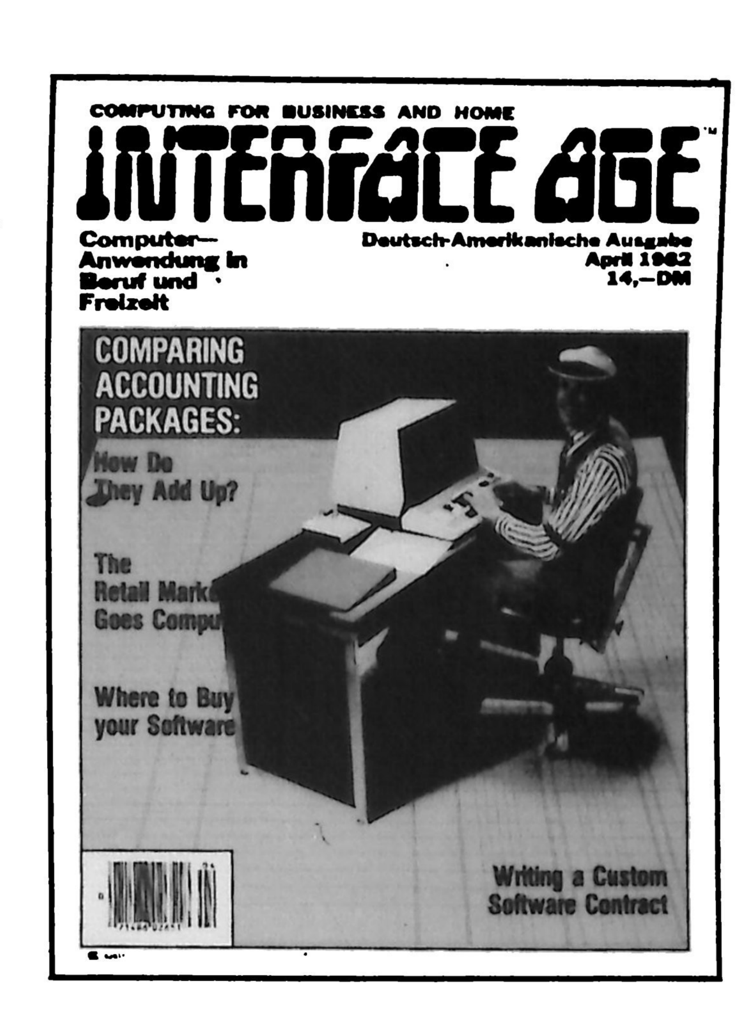
VARPTR 56
VERIFY 75
VERIFY* 75
VPLOT 29

WAIT 71
Warmstart 105

Ankündigung der deutsch-amerikanischen Ausgabe der Fachzeitschrift INTERFACE AGE

Der reichhaltige, deutschsprachige Teil des Magazins analysiert für Sie unter anderem wertvolle Hard- und Software, zeigt Ihnen an Beispielen die Nützlichkeit von Mikrocomputern im geschäftlichen Bereich und stellt Ihnen jüngste Erzeugnisse der Mikrocomputerindustrie vor.

Der englischsprachige Teil liefert Ihnen ausführlichste Vergleichstabellen von Hard- und Software und berichtet über das Neueste vom Neuen auf dem amerikanischen Markt.



Wir halten Sie über die Entwicklungen auf beiden Seiten des Atlantiks auf dem Laufenden. Abonnieren Sie heute!

Jahresabonnement per Luftpost DM 120,—
Jahresabonnement bei Versand per Schiff DM 80,—
Senden Sie Ihre Bestellung an; INTERFACE AGE
Vohburgerstr.1, 8000 München

UNTERNEHMENSBERATUNG ANDREAS DRIPKE

präsentiert

EXBASIC LEVEL II

Ein Produkt von Michael Krause und Andreas Dripke

im weltweiten Vertrieb von

INTERFACE AGE

EXBASIC LEVEL II stellt ein stark erweitertes Basic für Commodore Computer dar. Insgesamt stehen über 75 neue, äußerst leistungsfähige Funktionen zur Verfügung. Die Implementierung für CBM-Computer erfolgt in zwei 4 kByte Eproms, die einfach in zwei freie Sockel des Computers eingesteckt werden. Zu EXBASIC LEVEL II erhalten Sie ein ausführliches, 120 Seiten starkes Anleitungsbuch mit Einbauanweisung, LEVEL II BASIC - Kurs und zahlreichen Beispielen.

Hilfsfunktionen:
FIND, AUTO, DEL, RENUM, TRACE (OFF), ON/OFF, DUMP, MATRIX, LETTER (OFF), FAST (OFF), STOP ON/OFF, MEM, HIMEM, ".", SPACE (OFF), GO, HELP, HELP*, BASIC, MERGE, MERGE*.

Graphikbefehle:
PRINT AT, HPLOT (320/640 x 25), VPLOT (200 x 40/80), SPACE, SET, RESET, POINT.

LEVEL II BASIC:
IF..THEN..ELSE, RESTORE Zeilennummer, ON..RESTORE Zeilennummer, REK, DISPOSE NEXT, DISPOSE RETURN, DISPOSE CLR, INPUTLINE, INPUTFORM, DEF USR=, DEF CALL=, CALL, DOKE, DEEK, VARPTR, SPACE, STRING$, INSTR, EVAL, EXEC, SWAP, SEC, BEEP, PRINT USING, PRINT USING # ON ERROR GOTO, RESUME, RESUME NEXT, RESUME Zeilennummer, WAIT-STOP, "'", HARDCOPY.

Zusätzlich bietet EXBASIC LEVEL II (je nach Serie):

2001/3001/4001: DOS 1.0 Support, Kassettenoperationen mit fünffacher Geschwindigkeit, MOD.
8001: Bildschirmsonderbefehle, Assembler/Disassembler.

EXBASIC LEVEL II ist erweiterbar mit SOFTMODULEN. Standard-SOFTMODULE (werden mitgeliefert): SORT (sortiert eindim. Variablenfeld in Sekunden), CLEAR (löscht Variablenfeld), GOTO X, GOSUB X. Weitere SOFTMODULE erhöhen die Leistungsfähigkeit von EXBASIC LEVEL II noch um ein Vielfaches.

UNTERNEHMENSBERATUNG ANDREAS DRIPKE

Vertrieb: INTERFACE AGE Verlagsgesellschaft m.b.H.
Vohburgerstr. 1, D-8000 München 21, Tel. 089/5806702

Betrifft: **Perfekter Kopierschutz für Ihre Programme**

Sehr geehrte Damen und Herren,

wenn Sie Softwareanbieter sind, wissen Sie um die Probleme mit Software-Kriminalität - Raubkopieren. Der Branche gehen Jahr für Jahr immense Summen verloren - auch Ihnen! Nach Schätzungen von Experten handelt es sich bei über 50 % aller im Umlauf vorhandenen Programmexemplare um illegale Kopien!

Wir bieten Ihnen jetzt ein perfektes Schutzsystem für Ihre Commodore-Software. Mit unserem Schutzverfahren gesicherte Programme können weder aufgelistet noch verändert und natürlich erst recht nicht kopiert werden. Sie erhalten eine persönliche elektronische Kennung, die praktisch nicht zu "knacken" ist.

Sorgen Sie dafür, daß Ihre wertvollen Programme nicht länger in Form von Raubkopien im Umlauf sind. Und erhöhen Sie auf diese Weise die Wirtschaftlichkeit Ihres Unternehmens - Sie sollten handeln, und zwar jetzt!

Fordern Sie unsere Information "Schutzpakete" an!

Professionelles Assembler Entwicklungs- und Lehrsystem für alle Commodore:

T.EX.AS.

"Terminal Extended Assembler" von Ralph Babel und Andreas Dripke

Assembler - Der Traum jedes Basic-Programmierers

Die Programmiersprache Assembler zu beherrschen bedeutet Zugriff zu haben zum "Herz" des Computers. Kein Wunder also, daß die besten Programme in Assembler geschrieben sind. Wann programmieren Sie in Assembler?

Jetzt können Sie Assembler einfach und ohne Vorkenntnisse lernen

'T.EX.AS.' vereint ein professionelles Assembler Entwicklungssystem mit einem für jedermann verständlichen Assembler-Kurs. Durch diese einzigartige Mischung aus Theorie und Praxis lernen Sie besonders schnell, umfassend und praxis-orientiert. Sie benötigen dazu wirklich keinerlei Vorkenntnisse in Assembler, unser Lehrbuch zeigt Ihnen von Anfang an den richtigen Weg.

T.EX.AS. ist unkompliziert und komfortabel zu handhaben

Die Implementierung von 'T.EX.AS.' erfolgt durch einfaches Einstecken. Zur Aktivierung genügt ein einziger SYS-Befehl. Assembler-Programme können so einfach wie Basic-Programme erstellt werden. 'T.EX.AS.' arbeitet sowohl mit einem Kassettenrekorder als auch mit einer Floppydisk zusammen.

T.EX.AS. - Entwicklungs- und Lehrsystem in Einem

Wenn Sie Assembler mehr und mehr beherrschen, werden Sie feststellen, daß Sie mit 'T.EX.AS.' gleichzeitig ein professionelles Entwicklungssystem besitzen, das Ihnen auch nach der Ausbildungsphase hervorragende Dienste leisten wird. Sie arbeiten also weiter auf dem System, auf dem Sie gelernt haben - das ist ideal!

T.EX.AS. arbeitet mit der international genormten Standardnotation

Zur Erhöhung der Schreibgeschwindigkeit können Sie alle Eingaben auch abkürzen. So genügt es z.B., wenn Sie "L(200X" eintippen, um "LDA (200,X)" einzugeben. Sie haben die Wahl zwischen dezimaler und hexadezimaler Zahlendarstellung. Und auch sonst bietet Ihnen 'T.EX.AS.' einen weit über das Übliche hinausgehenden Komfort. Sie dürfen von uns mit Recht etwas Besonderes erwarten.

T.EX.AS. - Professionelle Qualität

Die Möglichkeiten von 'T.EX.AS.' sind frappierend. Selbst der erfahrene Assembler-Programmierer wird die herausragenden Funktionen von 'T.EX.AS.' mit angenehmem Erstaunen zur Kenntnis nehmen. So haben Sie gleichzeitig einen Makro-Assembler und einen Line-by-Line-Assembler. Der Bildschirm kann in bis zu vier unabhängige Teilbereiche gesplittet werden. Sie haben komplexe Möglichkeiten zum Tracen von Maschinenprogrammen mit Einblendung von Testregistern, Breakpoint-Handling, Transfer-, Change und Update-Kommandos, eine Find-Funktion mit einem geradezu unglaublichen Pattern-Matching und vieles andere mehr. Sie werden staunen, was wir Ihnen zu bieten haben!

Fragen Sie Ihren Commodore Händler nach T.EX.AS. - oder fragen Sie uns!

Der Preis? Nur DM 298,-! Lieferbar ab ca. Ende '82!

Auslieferung in der Reihenfolge der Bestelleingänge.